Hardware Looper

Borja Morcillo Salgado

Trabajo de fin de grado

Grado en Ingeniería de Computadores
Facultad de Informática

Universidad Complutense de Madrid

Madrid, junio de 2020

Directores:

José Manuel Mendías Cuadros
Carlos González Calvo

Autorización de difusión

El abajo firmante, matriculado en el Grado de Ingeniería de Computadores de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Grado: "Hardware Looper", realizado durante el curso académico 2019-2020 bajo la dirección de Carlos González Calvo y José Manuel Mendías Cuadros del Departamento de Arquitectura de Computadores y Automática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Borja Morcillo Salgado

Madrid, a 30 de mayo de 2020

Agradecimientos

Esta es la típica página de agradecimientos, así que en un ramalazo de ego voy a hablar de lo personal.

Gracias a Mendi, por incitarme a la locura en el fabuloso mundo del *hardware*; y a Carlos, por apuntarse a capitanear el barco en esta trepidante aventura.

Gracias a la gente de computadores (y alguno que otro de software, informática y filosofía), que son majetes y amenizan mis días en la enorme caja de ladrillo visto.

Gracias a Chiqui y Ana, por traerme a sufrir al pedrusco (y regalarme la guitarra para mitigarlo). Y también a Vera, aunque no pinta mucho.

Gracias al bar El Tropezón y las criaturas que allí habitan, por motivos incompatibles con la formalidad que rige esta memoria.

Gracias a quien inventase la música, a los grupos, a Radio 3, a la gente que hace las cosas por auténtico amor al arte.

Gracias a Ana (no la misma de antes). No sé por qué, pero no deja de echarme cables. ¿Acaso he hecho algo bien?

Índice general

Índice de figuras

Índice de tablas

Resumen

Este proyecto consiste en el diseño e implementación íntegramente en hardware de un dispositivo digital destinado a la creación de música en directo, capaz de grabar fragmentos de sonido y reproducirlos en bucle. El sistema construido dispone de múltiples pistas para grabación, que son sincronizadas entre sí de forma automática. Además presenta varias entradas y salidas de audio de alta fidelidad (24 bits de resolución a 48 KHz), las cuales pueden ser conectadas según se desee. También es capaz de aplicar simultáneamente diversos efectos al sonido, encadenándolos en el orden que especifique el usuario; entre ellos *delay*, *overdrive*, *fuzz*, compresor, *flanger*, filtros o trémolo. Incluye salida de vídeo y puede ser controlado mediante un teclado o bien remotamente desde un *tablet* a través de un enlace *Bluetooth*.

Su arquitectura es altamente paralela con vistas a proporcionar el máximo rendimiento posible, así como una buena escalabilidad.

Palabras clave

Diseño hardware, FPGA, VHDL, Procesado digital de audio, Sonido, Tiempo real, Looper, Música, Música electrónica, Lo-fi

Abstract

This project involves design and implementation in hardware of a digital device aimed to create live music, able to record sound fragments and playing them in loop. The built system has multiple tracks for recording, automatically synchronized between them. It also has various high-fidelity audio inputs and outputs (24 bits resolution at 48 KHz), which can be connected as desired. Moreover, it is able to apply effects to the audio, with user configurable chain order; among others, includes delay, overdrive, fuzz, compressor, flanger, filters or tremolo. A video output is present too, and can be controlled with a keyboard or a remote tablet app using a Bluetooth connection.

Its architecture is highly parallel, focused on providing the highest performance as well as a good scalability.

Keywords

Hardware design, FPGA, VHDL, Digital audio processing, Sound, Real-time, Looper, Music, Electronic music, Lo-fi

Capítulo 1

Introducción

1.1. Motivación

Durante los últimos ciento cincuenta años, la música ha sufrido una serie de constantes cambios e innovaciones en todos sus aspectos. Empezando por las composiciones han surgido innumerables nuevos géneros, los cuales a su vez se han ido fusionando entre ellos. No cabe duda de que el abanico estilístico es infinito.

Por otro lado, la constante aparición de nuevos instrumentos y técnicas ha permitido ampliar inmensamente los horizontes de las obras musicales. La incursión del mundo digital en nuestras vidas ha repercutido en todo lo que nos rodea, y la música no es una excepción.

Los instrumentos convencionales, que podríamos considerar analógicos, quedan a la merced del intérprete; están sujetos a nuestras imperfecciones. Dichos defectos se consideran hermosos, hacen del sonido algo natural con la aleatoriedad que ello conlleva. No obstante, poder hacer música con ordenadores y generar ondas sonoras sintéticas ha facilitado que cualquier persona pueda desarrollar su creatividad, delegando en una máquina la destreza y conocimientos que exigen los instrumentos. Para sonar en la radio ya no es necesario siquiera un grupo de tres, cuatro o las personas que se requieran; armados con guitarras y bajos. Basta un ordenador, y quizá un micrófono.

Aquí es donde entra en juego este proyecto. Una sola persona haciendo música en tiempo real, sin necesitar una formación previa, mientras combina los mundos analógico y digital.

En resumidas cuentas, la idea es poder explotar la técnica del *live looping*[1], haciendo uso de un solo dispositivo que integre toda la funcionalidad necesaria para que una única persona suene como un grupo entero.

No obstante, el título de este trabajo de fin de grado es *Hardware Looper*, no solo hay motivos para crear un *looper*, sino también para hacerlo en *hardware*.

Implementar el sistema en una FPGA permite obtener un rendimiento mucho mayor del que se alcanzaría con *software* sobre microprocesadores o microcontroladores, a un coste económico menor. Por otro lado, dado que su *hardware* es dinámicamente reconfigurable, es posible actualizar el diseño en el futuro.

Además el *hardware* es inherentemente paralelo, lo cual facilita mantener sistemas de grandes prestaciones sin que ello afecte negativamente a su rendimiento.

Por tanto, crear un diseño *hardware* dedicado es una garantía de futuro, aunque en términos de trabajo y esfuerzo, requiere una mayor inversión inicial.

1.2. Objetivos

El principal objetivo del proyecto es diseñar e implementar sobre una FPGA, un dispositivo al cual conectar la salida de audio de instrumentos analógicos o digitales, siendo este capaz de grabar fragmentos para reproducirlos en bucle posteriormente.

Además, para mayor versatilidad, el sistema debe poder mezclar estas entradas (así como las salidas) permitiendo todo tipo de combinaciones. Por otro lado, el sonido ha de procesarse para aplicar diversos efectos a la salida.

La fusión de los mundos analógico y digital no es trivial, y aun menos cuando se trata de fragmentos grabados, ya que un humano no puede determinar con precisión absoluta la duración de los mismos. Esto conduce a problemas de alineamiento y sincronización de las grabaciones, que deben solventarse implementando mecanismos específicos.

Dado a la multitud de funciones y todas las opciones configurables, ha de elaborarse

[1]En el ámbito de la interpretación musical, la técnica conocida como *live looping* consiste en la grabación y posterior reproducción en bucle de fragmentos en tiempo real. Surge a mediados de los 90.

también un medio para controlar y visualizar el estado de la máquina. Este debe ser fácil de usar.

Respecto al diseño, la arquitectura del sistema debe ser modular y paralela, además de presentar una buena escalabilidad. Los hechos que motivan una implementación en *hardware* puro se encuentran en la sección 2.2.

1.3. Antecedentes

Ciertamente no es demasiado amplio el abanico de trabajos académicos similares a este *looper*, no obstante sí existen algunos dedicados a la implementación en *hardware* de procesadores de efectos en tiempo real. También existen artículos de investigación en la misma línea. [2–4]

Dichos trabajos e investigaciones coinciden en sus conclusiones respecto a la alta velocidad de procesamiento que se logra gracias a emplear FPGAs, así como en la buena escalabilidad que ofrece este tipo de plataforma.

Más allá del mundo académico existe una gran variedad de dispositivos comerciales con fines comunes a los de este proyecto (figura 1.1). Sin embargo, desde el punto de vista funcional no es nada habitual encontrar en ellos múltiples entradas y salidas de audio. Algunos sí son capaces de incorporar efectos al audio.

Desde la perspectiva técnica, todos estos productos utilizan microcontroladores para ejecutar un *software* que define su comportamiento. De hecho, en términos de procesamiento de audio digital, solo existe una empresa que comercialice dispositivos con implementaciones en *hardware* dinámicamente reconfigurable: *Antelope Audio, Inc.* Se trata de procesadores de efectos, como el de la figura 1.2. Esta clase de producto está enfocado únicamente a entornos profesionales donde se requieren las prestaciones más altas posibles y los presupuestos son notablemente amplios.

(a) *Ditto X4*.

(b) *Boss RC-505*.

(c) *Boss RC-300*.

Figura 1.1: *Loopers* multipista.

Figura 1.2: Procesador de efectos digital con FPGA (modelo *Discrete 8 Synergy Core*).

1.4. Plan de trabajo

El desarrollo de este proyecto ha abarcado desde septiembre de 2019 a junio de 2020. Para su organización se ha estimado conveniente establecer una serie de fases secuenciales, priorizando aquellas que más dependencias generaban. Por orden, pueden enumerarse las siguientes:

1. **Planteamiento del proyecto.**

En esta fase previa al desarrollo, se concretó la índole del sistema: un *looper* implementado en *hardware* sobre FPGA. A pesar de que el proyecto no comenzaría hasta meses más tarde, fue necesario para realizar la inscripción.

2. **Análisis de la plataforma escogida y valoración de opciones de implementación.** *(Dos semanas).*

   Una vez escogida la placa de prototipado a utilizar, se determinaron ciertas restricciones de diseño impuestas por sus componentes; tales como la anchura de las muestras de audio o profundidad de color del vídeo. Del mismo modo se comenzaron a concretar los detalles funcionales del sistema: grabación en múltiples pistas, efectos regulables, varias entradas y salidas de audio, etcétera.

3. **Desarrollo de un controlador de memoria DDR2 adaptado a la placa de prototipado.** *(Seis semanas).*

   En base a las conclusiones de la fase anterior, se consideró comenzar con una implementación puramente *hardware* del controlador de memoria. De este componente depende la grabación del sonido, algo imprescindible en un *looper*.

   Durante esta fase, partiendo de un controlador previo se elaboró uno nuevo ajustado a los requisitos del sistema, utilizando la frecuencia de reloj apropiada y la mayor anchura de palabra posible. Esto requirió además configurar debidamente un *IP soft core* que actúa como interfaz de memoria.

   Una vez fue posible escribir y leer todas las direcciones de la memoria, se plantearon varias arquitecturas para dar solución a los problemas de acceso concurrente, alineamiento de datos y tiempos de acceso. Cuando finalmente quedó diseñado el controlador, se procedió a su implementación en *hardware*.

   Tras el éxito en su verificación y corrección de errores, se descarta la opción de un co-diseño *hardware-software* (que facilitaría el acceso a memoria, implementando el controlador en *software*).

4. **Planteamiento general de la arquitectura.** *(Una semana).*

   Una vez resuelto el controlador DDR2 quedaron fijados parámetros como la frecuencia de reloj del sistema o el número de pistas. Se establecen también las líneas generales del diseño, su jerarquía y los principios de la comunicación entre módulos. Además fue necesario fijar qué componentes serían necesarios y qué tareas debían realizar para satisfacer la funcionalidad.

5. **Diseño del controlador I2S.** *(Una semana).*

   El controlador I2S es necesario para la entrada/salida de audio en el sistema, y por tanto para validar todos los módulos que atraviesa el sonido.

   A partir de un controlador diseñado para funcionar a 75 MHz, se elaboró uno nuevo compatible con el sistema de múltiples entradas y salidas.

   Al completar la implementación del mismo, fue posible probar las capacidades de grabación y reproducción del controlador de memoria diseñado previamente.

6. **Diseño de los mezcladores de entrada y salida.** *(Dos semanas).*

   Durante esta fase se construyeron los módulos que mezclan las múltiples entradas y salidas del sistema, acordes a la arquitectura planteada. Fue necesario diseñar un sistema paramétrico capaz de establecer dinámicamente todas las posibles combinaciones de rutas entre entradas, pistas y salidas de audio. También se abordaron los diversos problemas que surgen al mezclar el audio (como la saturación) mediante mecanismos de control.

7. **Diseño de las pistas y mecanismos de sincronización.** *(Cuatro semanas).*

   La implementación de las pistas supuso un nexo de unión entre la entrada/salida de audio y el acceso a memoria.

   Durante esta etapa del desarrollo se elaboró toda la lógica de control, así como los mecanismos de sincronización y coordinación entre pistas, puesto que en ellas se ad-

ministran las grabaciones. Además fue necesario diseñar el generador de direcciones de memoria (para la comunicación con el controlador DDR2), el cual también está sujeto a la sincronización.

Por otro lado también se implementó el metrónomo, dispositivo central para el control del *tempo* en el sistema.

8. **Integración de los controladores para VGA y PS/2**. *(Dos semanas).*

   Es estrictamente necesaria la presencia de estos controladores para el manejo de las pistas, por tanto fueron implementados inmediatamente después. El proceso consistió en adaptar un controlador de vídeo para hacerlo compatible con la placa de prototipado empleada; así como generar las estructuras combinacionales y los mapas de bits (almacenados en ROMs) necesarios para imprimir la información gráficamente en el monitor externo.

   En el caso del teclado fueron definidos los procesos de traducción de mensajes PS/2 a señales internas para el control del *looper*.

   Ambos controladores fueron encapsulados en sendos componentes destinados a concentrar en ellos las tareas de retransmisión del estado y recepción del control. De este modo es posible incluir otros controladores de interfaz humana tal que funcionen simultáneamente.

   Llegado este punto, se había alcanzado una versión preliminar del sistema.

9. **Diseño de los módulos de control por *Bluetooth*.** *(Tres semanas).*

   El siguiente objetivo fue diseñar los módulos que realizan la comunicación por *Bluetooth*, y por tanto establecer el protocolo y formato de las tramas de envío y recepción. No obstante, en primer lugar se comprobó que fuese posible la comunicación entre ambos dispositivos. Inicialmente hubo problemas de incompatibilidad, por lo que fue necesario cambiar de módulo *Bluetooth*.

Puesto que el tiempo de respuesta es crítico, se estimó conveniente minimizar tanto como fuese posible la cantidad de información intercambiada, a pesar del efecto negativo en la complejidad. De este modo únicamente se realizan envíos o recepciones del valor de un parámetro cuando este cambia, en lugar del estado de todo el sistema periódicamente.

Para el receptor fue preciso un intérprete de tramas, que genere las señales de control pertinentes. Sin embargo, para el emisor fue necesario un sistema adaptado a la lentitud de la retransmisiones a través de *Bluetooth*, que conllevan múltiples ciclos de reloj. Este consiste en un mecanismo de etiquetas y un generador de tramas encolables.

10. **Implementación *software* de la aplicación de control.** *(Tres semanas).*

    Para controlar el sistema a través de *Bluetooth* se desarrolló una aplicación destinada a dispositivos móviles con sistema operativo *iOS*. Durante el transcurso de esta etapa fue elaborada una arquitectura software capaz de realizar la comunicación para el envío, recepción y representación de los datos.

    Los eventos recibidos son interpretados, su información es almacenada y una interfaz gráfica animada la muestra por pantalla. Esta última es interactiva y por ello monitoriza las acciones del usuario, para así generar tramas de control y enviarlas al *looper*.

11. **Diseño del procesador de efectos.** *(Cuatro semanas).*

    El primer paso de esta fase fue diseñar el sistema que permite seleccionar en qué orden se deben aplicar los efectos.

    A continuación, el diseño de los efectos exigió en primer lugar realizar implementaciones *software* de los mismos y las pruebas de sonido pertinentes, para posteriormente implementar en *hardware* las arquitecturas correspondientes a los mismos.

    Los efectos tienen arquitecturas dispares entre sí: algunos utilizan memorias, diversos operadores aritméticos, elementos de lógica combinacional...

12. **Incorporación de filtros.** *(Dos semanas).*

    Para completar la colección de efectos ofrecida, fueron diseñados e incorporados unos filtros al procesador de efectos. Fue necesario investigar sobre los mismos, para escoger el tipo más conveniente y posteriormente implementarlo. Al igual que con los otros efectos se realizaron previamente simulaciones *software*.

13. **Elaboración de una carcasa para el *looper*.** *(Dos semanas).*

    Finalmente para montar todos los componentes del producto, fue diseñado un chasis utilizando herramientas de dibujo asistido por ordenador (CAD). Para materializar el mismo se hizo uso de una impresora 3D de modelado por deposición fundida.

14. **Frecuencia de corte variable en los filtros.** *(Una semana).*

    Se decidió mejorar la implementación de los filtros, adaptándolos para que fuese posible variar la frecuencia de corte. Después de valorar diversas alternativas, se procedió a completar el diseño.

1.5. Organización de esta memoria

El resto de esta memoria, está estructurado en una serie de capítulos cuyos contenidos quedan descritos a continuación:

- **Visión general del sistema LoopMAN**: Se describe qué es y qué puede hacer el dispositivo, así como las líneas generales de su estructura interna. También se incluyen los motivos que han impulsado a utilizar una FPGA como plataforma.
- **Contexto tecnológico**: Recopila todos los medios que han sido necesarios para llevar a cabo este proyecto. Contiene información técnica sobre los componentes del sistema, protocolos, estándares, algoritmos implementados y herramientas utilizadas.
- **Arquitectura hardware e implementación de LoopMAN**: Detalles de la implementación en *hardware* de los componentes en que se subdivide el sistema, así como de

sus módulos internos y la comunicación entre los mismos. Esto incluye los elementos del flujo principal del audio, así como los mecanismos de control, sincronización y la gestión de interacción entre el usuario y el *looper*.

- **Desarrollos complementarios: carcasa y *App* para el control remoto de *LoopMAN***: Aborda brevemente la funcionalidad que ofrece la *app* y su arquitectura interna. También contiene las características del chasis diseñado para albergar los componentes del sistema.

- **Resultados, conclusiones y trabajo futuro**: Propiedades del sistema final construido, así como aquello que se ha concluido durante el desarrollo del trabajo de fin de grado. También incluye distintas vías para posibles continuaciones del proyecto realizado.

- **Bibliografía**: Relación de fuentes referenciadas en la memoria (artículos, libros, trabajos previos...).

Capítulo 2

Visión general del sistema LoopMan

2.1. Funcionalidad

El principal cometido de este sistema, al que me referiré a lo largo del texto como *LoopMAN* (figura 2.1), es la grabación de fragmentos de audio para reproducirlos en bucle, de ahí que su nombre derive de *looper*. Sin embargo, sus capacidades van mucho más allá de las disponibles en *looper* convencional como el mostrado en la figura 2.2a. Seguidamente las detallaré.

2.1.1. Entrada de audio

Siguiendo el flujo del sonido por el interior de este sistema, la primera función disponible es la mezcla de las distintas entradas de audio.

Al igual que sucede con las mesas de mezclas (figura 2.2b), el objetivo es combinar los múltiples canales de entrada en uno solo, tal que distintas fuentes de entrada tengan la misma salida.

La diferencia es que en este sistema existen ocho mezcladores, para generar ocho salidas distintas, que resultan de combinar los canales de entrada tal y como se desee.

Cada una de dichas salidas queda directamente conectada a una pista de grabación. De esta manera puede unificarse el audio procedente de múltiples instrumentos en una única señal.

Figura 2.1: *LoopMAN*.

2.1.2. Pistas: grabación y reproducción

La grabación de sonido se realiza en las pistas. Las pistas son las entidades básicas del sistema, hay ocho. Estas indican al mezclador de entrada qué canales de entrada quieren recibir.

Las pistas realizan la función de *looper*, es decir, realizan la grabación y reproducción de la mezcla que reciben.

Al comenzar la grabación, el sonido se almacenará en la memoria. Por defecto, la primera grabación en una pista determina cuanto duran los fragmentos. También se puede extender o acortar la duración de un fragmento ya grabado, permitiendo añadir nuevas grabaciones a continuación o limitar la longitud de las existentes.

En caso de grabar un nuevo fragmento sobre otro existente, ambos se mezclarán. Esta función es conocida como *overdub* y es ajustable, siendo posible indicar cuánta presencia de la grabación anterior se desea mantener en la nueva.

Cuando una grabación es detenida, puede comenzar a reproducirse desde el principio

(a) *Looper* convencional. Una pulsación para grabar, otra para reproducir.

(b) Mesa de mezclas. En la parte superior, se observan las múltiples entradas.

Figura 2.2: *Looper* y mesa de mezclas

accionando la opción *"play"*. Si dicha opción estaba activa al finalizar la grabación, la reproducción comienza automáticamente.

Cabe mencionar que es posible comenzar a grabar automáticamente, mediante la función *"autorec"*. Al activarla, la pista monitorizará la amplitud del audio de entrada e iniciará la grabación cuando detecte que está sonando. Esto ayuda al músico a mantener la atención en el instrumento, evitando que se distraiga pensando en el momento de pulsar el botón, o aún peor, ocupando una de sus manos para realizar la pulsación.

Las pistas, envían continuamente a su salida la entrada que reciben. Mientras se esté reproduciendo una grabación, se mezclará con la entrada, tal que el sonido grabado y la entrada actual puedan sonar simultáneamente.

Además, cada pista tiene un control de volumen para regular su salida. La opción *"mute"* silencia por completo la salida de la pista. Existe además una opción *"solo"* para silenciar todas las demás pistas, de modo que cuando una o más pistas la activen, sean la únicas que puedan escucharse.

La opción *"clear"* elimina todo el audio grabado en la pista.

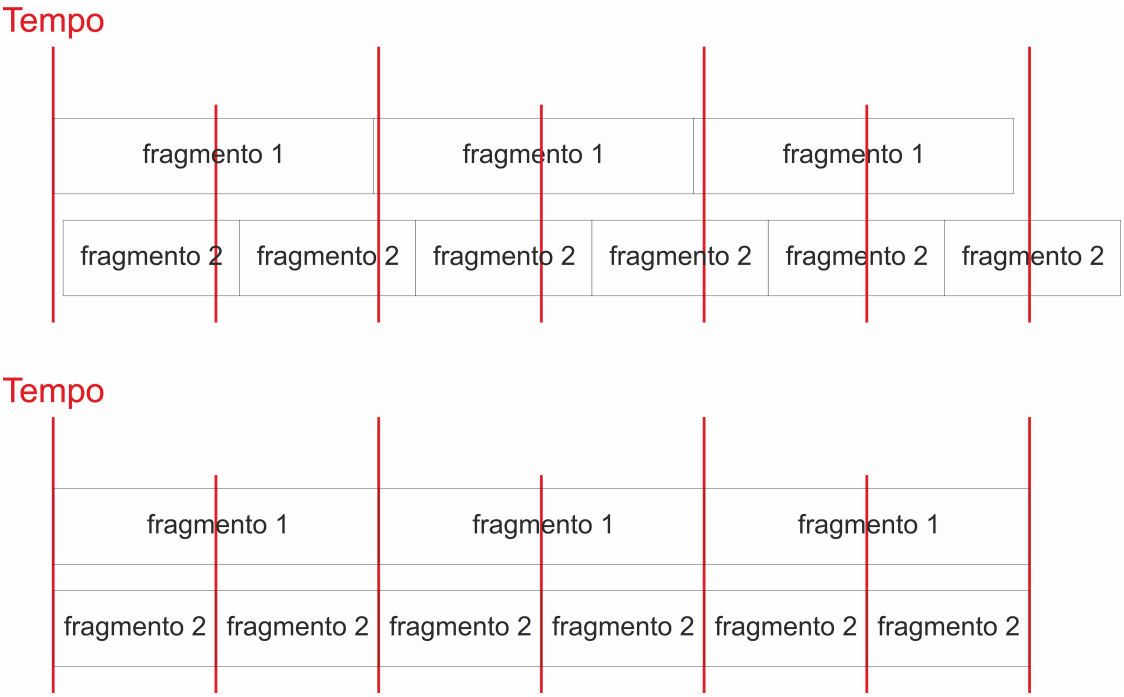


Figura 2.3: Pistas no sincronizadas arriba, pistas sincronizadas abajo. Se observa como cada vez es más perceptible la falta de sincronía.

2.1.3. Pistas: sincronización

La existencia de varias pistas en el sistema hace imprescindible un mecanismo de sincronización para garantizar que todas siguen el mismo *tempo*[1].

Cuando el intérprete graba varios fragmentos para su reproducción en bucle, es imposible que todos ellos duren lo mismo, o bien que exista una relación exacta entre los mismos (e.g. un cuarto, un octavo...).

Este problema ocasiona que tras múltiples reproducciones de ambos bucles, las diferencias de duración comiencen a acumularse, dando lugar a que una suene antes que la otra. Este efecto está representado gráficamente en la figura 2.3.

Este fenómeno queda solucionado en el sistema gracias al control centralizado del *tempo*. La velocidad es ajustada por el usuario, determinando este la cadencia que desee. Después, un reloj central genera pulsos con la periodicidad indicada y todas las pistas deben ajustarse

[1]En terminología musical, *tempo* hace referencia a la velocidad con la que debe ejecutarse una pieza.

a ellos.

De este modo se facilita enormemente sincronizar el *looper* con otros instrumentos electrónicos, por ejemplo, secuenciadores o cajas de ritmos (ver figura 2.4). También permite establecer rigurosamente el *tempo* en las piezas interpretadas por humanos.

Por ende, sin dicha medida se hace prácticamente imposible la fusión de los mundos analógico y digital.

(a) Secuenciador. Reproduce una secuencia programable de sonidos.

(b) Caja de ritmos. Es una batería electrónica capaz de funcionar automáticamente.

Figura 2.4: Secuenciador y caja de ritmos

Otro problema asociado a la sincronización es el instante en que comienzan las grabaciones y reproducciones. Si no hay ninguna pista sonando o grabando, en el preciso momento en que comience el proceso, el metrónomo central empezará a contar desde cero.

Si por el contrario existieran otras pistas activas al iniciar la grabación o reproducción, el sistema detecta el error producido por el músico y ajusta automáticamente el nuevo fragmento al *tempo* del sistema. En caso contrario, cuando el *loop* grabado comenzara a reproducirse, estaría desfasado respecto al resto de pistas.

Las funciones de sincronización pueden ser desactivadas si así lo decide el usuario.

Ajuste de tempo

Relacionado con la sincronización, cabe mencionar el funcionamiento del metrónomo del sistema. Existen dos vías para establecer el *tempo*.

Por un lado, mediante un ajuste manual tal que el usuario establezca un valor medido en bpm (pulsos por minuto). Por otro, a través de un pulsador se puede marcar la cadencia que se desee ver replicada automáticamente por el metrónomo. Es decir, que el usuario debe dar sucesivos toques en un botón, al ritmo del *tempo* que quiera. Con este método, las pulsaciones son monitorizadas para así calcular la frecuencia de las mismas.

2.1.4. Efectos

Las pistas envían su salida a un procesador de efectos. Estos módulos son capaces de aplicar múltiples efectos al audio en tiempo real, en el orden que especifique el usuario. Es decir, que el sistema posibilita indicar qué efectos han de ser aplicados primero y cuales después. Para mayor versatilidad, la ordenación es independiente en cada una de las pistas.

No es trivial el orden en que son aplicados los efectos al sonido, y cada músico tiene sus preferencias personales (figura 2.5).

Los efectos son regulables, pueden ajustarse los parámetros que rigen su comportamiento.

Cabe mencionar que cada efecto puede activarse o desactivarse incluso para el sonido ya grabado, ampliando las posibilidades del sistema. Al ser alterado en tiempo real, un mismo fragmento puede sonar de distintas formas.

A continuación se describe brevemente el comportamiento de los efectos disponibles.

Delay (eco)

Este efecto imita el eco, repitiendo el sonido cada cierto tiempo, pero cada vez con mayor atenuación.

La aplicación del *delay* produce las sensaciones de tener múltiples instrumentos tocando a la vez, de que las notas suenan varias veces, o de que el sonido se produce en un espacio con

Figura 2.5: El orden de los pedales sí altera el resultado.

eco. Depende del tiempo que tarda en repetirse el sonido, que es configurable en el efecto.

Este efecto es muy habitual en la música contemporánea (véase figura 2.6).

Trémolo

Este efecto simplemente aumenta y reduce cíclicamente la amplitud de onda del sonido.

Dada su sencillez, es uno de los efectos más antiguos, ya existía en la década de 1950 (ver figura 2.7). No obstante, el trémolo, también conocido como *vibrato*, es una técnica que se ha aplicado desde tiempos remotos en instrumentos de cuerda frotada: violines, violonchelos, contrabajos...

En el sistema es configurable la frecuencia a la que cambia la intensidad, es decir, a qué velocidad varía el volumen.

Jamón

El nombre de este efecto, "Jamón", se debe a que el sonido que produce es parecido a los órganos *Hammond* con altavoces rotatorios (también conocidos como *Leslie*). Se puede

Figura 2.6: Pedal de *delay*. El pulsador lo activa o desactiva, con el potenciómetro se ajusta la duración.

Figura 2.7: Pedal trémolo de 1950.

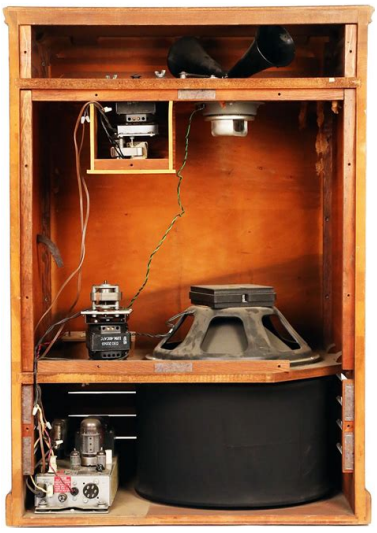


Figura 2.8: Altavoz rotatorio *Leslie*. En la parte superior hay dos bocinas (*tweeters*), en la inferior un *subwoofer* para los graves. Ambos giran.

observar en la figura 2.8, la presencia de dos altavoces uno arriba y otro abajo. Ambos giran mientras suenan, tal que se produce el efecto Doppler[2], alterándose cíclicamente el tono. También varía el volumen, de un modo similar al trémolo.

En el efecto, la velocidad de rotación es ajustable en tiempo real, tal que cambia la duración del ciclo.

Compresor

El compresor permite reducir el rango dinámico, disminuyendo el volumen de los sonidos de mayor intensidad, para equipararlos a los que suenan más bajo.

Una de las aplicaciones de este efecto es para prevenir la saturación de las muestras, tal que no se produzca distorsión cuando hay picos de intensidad.

Sin embargo, es muy habitual aplicar compresión para hacer más constante el sonido y que el oyente mantenga en él su atención, así se logra mayor claridad, especialmente en las vocales (ya sea en locuciones radiofónicas o en canciones).

Otro uso de la compresión es para aumentar la resonancia en instrumentos de percusión.

[2]El efecto Doppler consiste en el cambio de frecuencia aparente de una onda producido por el movimiento relativo de la fuente respecto a un punto de referencia.

Es posible configurar el umbral, el valor a partir del cual se aplica la compresión en la onda de salida.

Overdrive

Ampliamente utilizado en el *Rock and Roll* y todos sus sucedáneos más modernos, este efecto consiste en distorsionar intencionadamente el sonido (véase figura 2.9).

Para ello se limita la forma de onda, reemplazando por un valor constante los fragmentos de la misma que superan un cierto umbral. Típicamente, el valor de reemplazo y el umbral, coinciden.

El fenómeno que pretende imitar el efecto, es la saturación que se produce en los amplificadores cuando la ganancia es demasiado alta y las válvulas de vacío (o transistores), incapaces de generar la salida deseada, empiezan a distorsionar.

La intensidad del efecto es configurable en tiempo real.

Fuzz

El *fuzz* puede recordar al *overdrive* en ciertos casos, pero su efecto es mucho más intenso. Por ello se suele utilizar en conjunto con "puertas de ruido", para eliminar el zumbido constante que en ocasiones se produce.

Aplica una limitación asimétrica a la forma de onda, distorsionando en gran medida el sonido y alargando considerablemente la duración de las notas. Además provoca la aparición de armónicos, especialmente de los pares.

Al igual que en todos los efectos del sistema, se puede ajustar la intensidad del *fuzz*.

Flanger

Este efecto, se basa en mezclar la señal original con otra de retardo variable. Produce un sonido oscilante, similar al de la turbina de un avión. Es bastante utilizado en el *rock* clásico y especialmente en la música pop.

(a) Pedal de *overdrive*

(b) Pedal de *fuzz*

Figura 2.9: Pedales de *overdrive y fuzz*.

En la figura 2.10 se muestra un ejemplo de implementación comercial del *flanger*, en formato pedal.

En este sistema, frecuencia de oscilación es ajustable en tiempo real.

Filtros

Los filtros permiten ecualizar el sonido, haciendo posible eliminar frecuencias a partir de un valor de corte.

Este *looper* presenta dos tipos de filtro (paso alta y paso baja) implementados como efectos, tal que sea posible escoger en que orden son aplicados, y así atenuar (o no) frecuencias originadas por la acción de otros efectos. Por ejemplo si se desea eliminar los agudos de un bajo eléctrico, pero mantener los armónicos que aparecen al someterlo al efecto de *fuzz*.

El filtro paso baja elimina las frecuencias altas (las más agudas) a partir del valor seleccionado por el usuario. De forma completamente análoga, el filtro paso alta hace lo propio para las frecuencias bajas (es decir, las graves).

Posibles utilidades de estos, son por ejemplo, reducir el ruido de alta frecuencia, eliminar armónicos demasiado agudos o restarle protagonismo a una melodía.

La figura 2.11 presenta ejemplos de dispositivos con la misma función que la integrada en este sistema.

Figura 2.10: Pedal de *flanger*.

2.1.5. Salida de audio

La etapa final del flujo del sonido es el mezclador de salida. La función de este módulo es elaborar para cada salida de audio del sistema, la mezcla correspondiente.

De este modo, al igual que es posible seleccionar qué entradas recibe cada pista; se permite configurar por qué salidas sonará cada una.

Además, cada salida dispone de un regulador de volumen (*master*) y un compresor. Este último permite normalizar el sonido y eliminar la saturación.

Por poner un ejemplo, de este modo es posible tener una guitarra saliendo por un par de amplificadores, un micrófono y una caja de ritmos sonando a través de otro altavoz, y todo el conjunto conectado a un *subwoofer*[3] central. Incluso si este sistema careciese de su principal función (el *looper*), ofrecería una buena plataforma para gestionar múltiples fuentes y salidas de audio en tiempo real.

[3]El *subwoofer* es un tipo de altavoz diseñado específicamente para la reproducción de frecuencias graves, típicamente entre 20 y 80 Hz.

(a) Filtro paso baja.

(b) Filtro paso alta.

Figura 2.11: Filtros.

2.1.6. Control del sistema

Con vistas a facilitar el manejo del *looper*, el sistema incorpora dos mecanismos para controlarlo: un control directo mediante un teclado USB, y un control remoto (por *Bluetooth*) desde una aplicación *software* para *tablets*.

El teclado USB está pensado para configuraciones de menor presupuesto o entornos donde no hay cabida para un dispositivo móvil. Cada tecla tiene una función asignada para realizar el control de la reproducción, grabación, asignación de entradas y salidas... etcétera. Sin embargo, esta modalidad tiene una limitación: no todas las funciones son accesibles directamente (pues para ello sería necesario un teclado mucho más grande que el estándar), en algunos casos es necesario seleccionar previamente el elemento que va a alterarse. Esto sucede por ejemplo, con el volumen de las pistas, primero hay que seleccionarla y a continuación proceder al ajuste.

No obstante, si se dispone de los medios adecuados, es posible controlar el sistema me-

diante una aplicación *software* específicamente diseñada para este propósito. La conexión inalámbrica, así como la interfaz táctil de la *tablet*, agilizan enormemente el control. En la pantalla se muestran todos los controles disponibles, para reducir al máximo el tiempo que el usuario emplea en la interacción.

En cualquier caso, no hay ninguna restricción que impida utilizar ambos controladores al mismo tiempo, y siempre puede hacerse como medida de seguridad. Esta redundancia podría evitar grandes problemas, al tratarse de un sistema de tiempo real cuyo funcionamiento es moderadamente crítico en ciertas ocasiones; en concreto, durante una actuación musical en directo.

2.1.7. Visualización del estado

Del mismo modo que sucede con el control, existen dos vías para visualizar el estado. Por un lado mediante un monitor VGA, por otro, mediante la misma aplicación *software* empleada para el control.

Ambas modalidades muestran información sobre el estado de las pistas (si están grabando, reproduciendo, su longitud, volumen...), el metrónomo del sistema o los mezcladores de salida. Sin embargo, la aplicación *software* es más versátil y muestra información adicional sobre efectos, rutas de entrada/salida o determinados parámetros de las pistas.

De forma análoga al control, también pueden utilizarse conjuntamente el visor remoto en la *tablet* y el monitor. La figura 2.12 ilustra cómo el *LoopMAN* puede ser controlado mediante ambas vías de forma simultánea.

2.2. Por qué en una FPGA

Los sistemas de tiempo real necesitan tiempos de respuesta tan pequeños como sea posible, y este no es una excepción. El sonido es rápido, viaja a 340 m/s y cuando una parte de él rebota en alguna superficie, llega un poco más tarde a nuestros oídos: se trata de la reverberación. Este fenómeno es perceptible, como tantos otros, por ello en este ámbito no

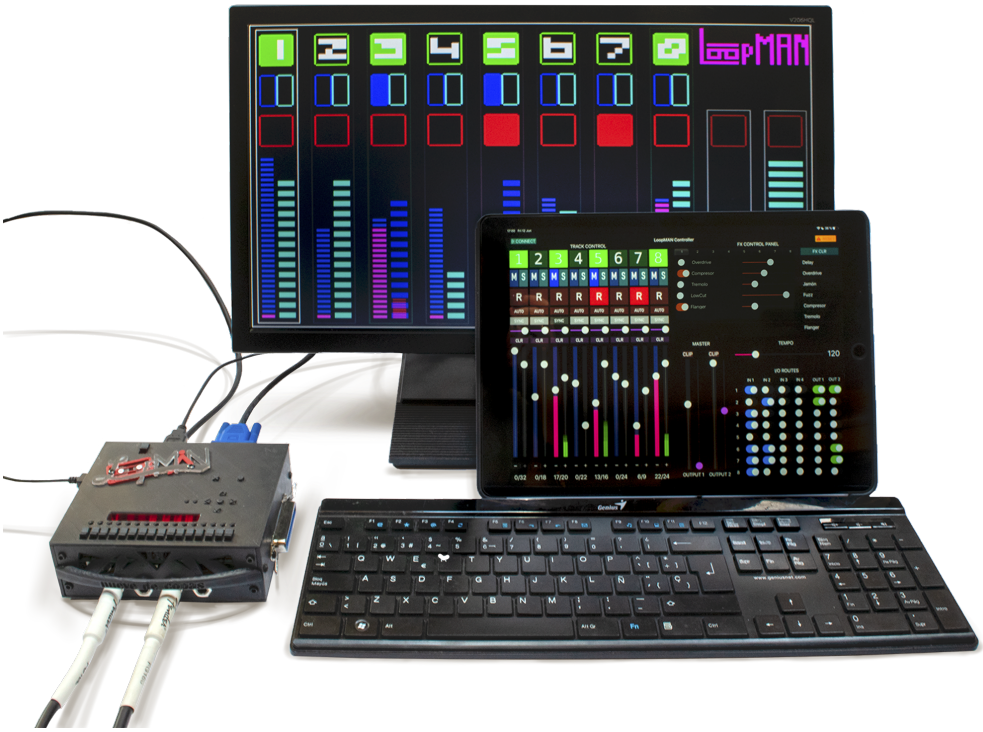


Figura 2.12: *LoopMAN* conectado a un teclado, un monitor y a la aplicación *Bluetooh*.

hay cabida para un sistema que al procesar induzca algún tipo de retardo.

También se ha de tener en cuenta que la índole del sistema es moderadamente crítica. No serviría en absoluto si se produce un fallo en mitad de una actuación musical, la tolerancia a errores ha de ser alta.

Por otro lado, muchas de las operaciones necesarias para el funcionamiento de este sistema son complejas desde el punto de vista computacional. Múltiples entradas mezclándose en múltiples pistas, siendo procesadas y grabadas continuamente, de forma paralela. Todo ello sin contar con los mecanismos de control y sincronización que requieren estar siempre activos.

Por tanto, se trata de una posición muy delicada para los sistemas *software*, cuyo diseño requiere de un sistema operativo corriendo por debajo. La infinidad de capas que hay que atravesar desde la entrada/salida hasta la aplicación supondría un altísimo sobrecoste, además de la inmensa cantidad de tareas e hilos de ejecución que deberían ser planificadas y sincronizadas. Aun con microprocesadores potentes capaces de desempeñar todas las tareas, la complejidad total del sistema lo hace mucho más susceptible de fallar.

Aunque sería más conveniente que el ejemplo anterior, tampoco es una solución prescindir del sistema operativo y construir una aplicación *bare-metal*. Esto supondría una enorme cantidad de trabajo al no disponer de los mecanismos de planificación que tanto facilitan la programación de alto nivel. Igualmente podría conllevar problemas de rendimiento.

Por ello se concluye que la mejor manera de implementar este sistema es directamente en *hardware*. De este modo se evita todo el sobrecoste antes expuesto, además de que cada componente funciona inherentemente en paralelo; así el rendimiento es máximo.

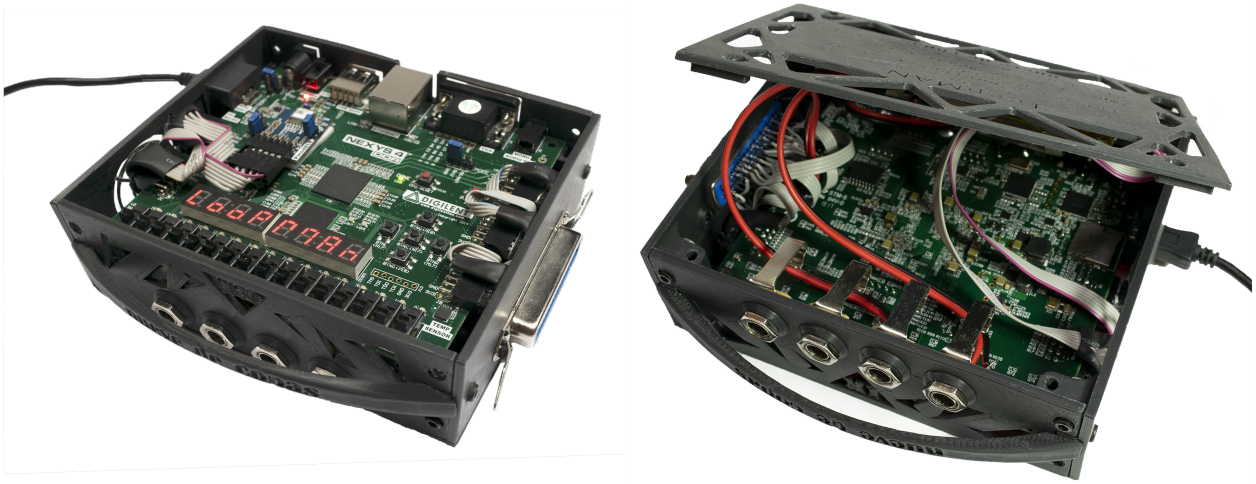
Para materializar la implementación, utilizar una FPGA permite elaborar innumerables prototipos, facilitando mucho las tareas de diseño.

Poder especificar el diseño a nivel RTL utilizando un lenguaje HDL (VHDL en este caso), es otra de las ventajas de esta tecnología. Gracias a los avances de las herramientas EDA, los procesos de síntesis realizan optimizaciones en la lógica de los diseños, proporcionando

resultados óptimos.

Además, la relación entre rendimiento y coste económico es mucho mejor de lo que se podría obtener con una implementación en *software*.

La figura 2.13 muestra la placa de prototipado (*Nexys 4 DDR*) con FPGA integrada en el interior del *LoopMAN*.



(a) Vista superior.

(b) Vista inferior.

Figura 2.13: Interior del *LoopMAN*.

2.3. Arquitectura (visión general)

Este proyecto consiste, principalmente, en un diseño *hardware* de propósito específico. La arquitectura está estructurada por componentes que funcionan en paralelo y se comunican entre ellos.

El diseño de la misma se ha elaborado con vistas a proporcionar una salida en el menor tiempo posible, sin renunciar a una buena escalabilidad.

Dando una perspectiva general, el flujo del sonido a través del sistema recorre, por orden: los mezcladores de entrada, las pistas, los procesadores de efectos y por último los mezcladores de salida. Cada componente realiza las operaciones pertinentes sobre la muestra de audio que recibe, una vez ha finalizado notifica al siguiente la transferencia del resultado

y así sucesivamente hasta la salida.

Por otro lado los mecanismos de sincronización, control y visualización del estado intercambian información con los otros componentes para regular su comportamiento y conocer su estado actual.

En la figura 2.14 se proporciona un esquema general de la arquitectura.

De los componentes mostrados en el esquema, el primero es *audioIO*. Este se encarga de recibir en paralelo las muestras a serializar, para enviarlas al códec de audio. Por otro lado, también recibe en serie el sonido entrante, para transmitírselo al siguiente módulo en el flujo del audio. En concreto al *trackInputMixer*, componente al que cada pista indica qué canales de entrada desea recibir, para que este realice la mezcla pertinente y se la envíe.

El sonido mezclado pasa a *trackController*, el núcleo central del sistema. Alberga las ocho pistas y el controlador de memoria. Por ello, este módulo recibe la información del *tempo* (procedente de *tempoGenerator*) para las labores de sincronización; además comunica el estado actual de las pistas y recibe las señales de control pertinentes.

El siguiente componente en la cadena es el controlador de efectos (*fxController*), que contiene los ocho procesadores de efectos, uno por pista.

Finalmente, *outputMixer* recibe desde cada pista a qué canales de salida debe ser enviado su audio, y realiza la mezcla correspondiente para conducirla a *audioIO*.

Los últimos tres módulos explicados: *trackController*, *fxController* y *outputMixer*; mantienen una información de estado y reciben señales de control. Esto permite la interacción con el usuario, la cuál se realiza mediante otros dos componentes: *userDisplay* y *userControl*, descritos a continuación.

UserDisplay recibe toda la información de estado del sistema, y la retransmite por dos protocolos: *Bluetooth* (para uso con una aplicación *software* específica) y VGA (para uso con un monitor externo).

Por otro lado, *userControl* es capaz de recibir, mediante *Bluetooth* (desde la aplicación *software*) o bien, un teclado USB, los datos de control del sistema. La información es identi-

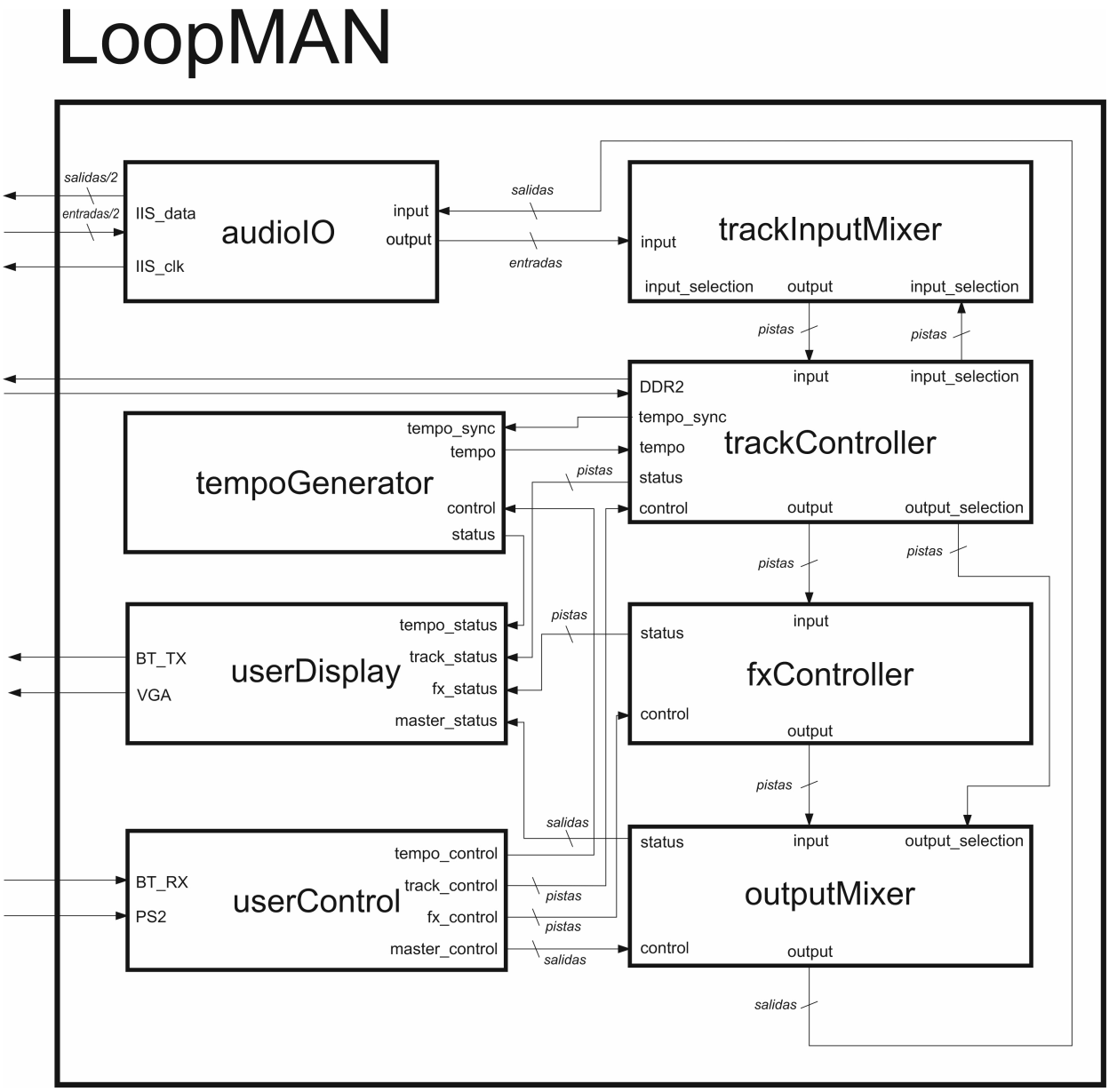


Figura 2.14: Módulo principal

ficada y dirigida a los módulos correspondientes, para que apliquen las acciones pertinentes.

Capítulo 3

Contexto tecnológico

Este capítulo describe los algoritmos y técnicas empleadas para proporcionar soluciones a los diversos problemas que se han presentado durante el desarrollo del proyecto.

También son abordados los detalles de los componentes utilizados en la construcción del sistema final; así como qué herramientas *hardware* y *software* que han sido necesarias a lo largo del proyecto.

3.1. Algoritmos y fundamento teórico

3.1.1. Cuantificación del audio

Es imprescindible obtener una representación digital de las ondas de sonido para poder operar con ellas, proceso que no es trivial si se consideran las dos dimensiones presentes: la amplitud y el tiempo.

Por un lado, mediante un conversor analógico-digital es posible obtener un valor digital aproximado de la amplitud de onda para un cierto instante. El rigor con que dicho valor refleja la realidad es directamente proporcional al número de bits que emplea para representar la amplitud. En formato CD (*Compact Disc*) se utilizan 16 bits, sin embargo en los estándares de alta fidelidad el número asciende a 24.

Para digitalizar la segunda dimensión, el tiempo, se recurre a un proceso conocido como muestreo. Consiste en la obtención periódica de muestras de sonido, tal que cada una queda

asociada a una porción de tiempo. Como cabría esperar, cuanto menor sea el periodo entre muestras tomadas, más precisos serán los datos. En el estándar CD se utiliza una frecuencia de muestreo de 44100 Hz. Aunque en alta fidelidad se llega a los 192000 Hz, esto excede con creces las bases teóricas del teorema de Nyquist-Shannon. Este último enuncia que para lograr la reconstrucción perfecta de una onda muestreada, la frecuencia más alta presente en la onda no debe ser superior a la mitad de la frecuencia de muestreo. Considerando que el oído humano es incapaz de percibir sonidos cuya frecuencia exceda los 20000 Hz, muestreando a más de 40000 Hz no se obtendría ninguna mejora apreciable en lo que respecta a la fidelidad del sonido.

En este sistema la entrada/salida de audio funciona con resolución de 24 bits, muestreados a 48828,125 Hz. Sin embargo, internamente trabaja a 32 bits por las razones que explicaré más adelante.

3.1.2. Punto fijo

Cuando se trata de operaciones aritméticas con valores no enteros, el procesamiento se complica en términos computacionales. En microprocesadores de propósito general (que disponen de unidades aritméticas avanzadas o coprocesadores numéricos), es frecuente recurrir al punto flotante para representar y operar con los valores. Sin embargo la complejidad técnica y el número de ciclos relativamente elevado que exigen estas operaciones, hacen de ello algo inviable para un sistema diseñado enteramente en *hardware* como este.

Por ello la aproximación más conveniente, aunque menos precisa, son las representaciones en punto fijo. La principal diferencia respecto al punto flotante es la posición de la coma decimal, que en este caso tiene una posición fija (como el propio nombre indica).

Por tanto, en este tipo de aritmética se destina un número prefijado de los bits menos significativos a representar el valor decimal, cuantos más bits sean, mayor será la precisión.

En otras palabras, al utilizar más bits pueden representarse más valores entre 0 y 1 (es decir, decimales), y por tanto menor será la distancia entre ellos. Como consecuencia

también será menor la diferencia entre el valor real y el representado, es decir, el error.

A pesar de la pérdida de precisión, el rendimiento que se obtiene al operar en puto fijo es mucho mayor, siendo posible realizar los cálculos en el mismo tiempo que si se tratase de valores enteros. Además la implementación es mucho menos costosa y consume menos recursos que el punto flotante. Otra ventaja es que es fácil operar con valores que tengan el punto decimal en distinta posición, aunque requiere realizar desplazamientos para alinear los operandos (en caso de la suma) o el resultado (en la multiplicación).

En un sistema que opera con muestras de audio es imprescindible trabajar con valores decimales y este *looper* no es una excepción y por los motivos recién expuestos, se ha decidido utilizar punto fijo en la aritmética. En primer lugar, las muestras de audio están escaladas en el rango [-1, 1], por lo que se representan necesariamente con valores decimales. Además en el sonido son muy frecuentes las operaciones que implican decimales, por ejemplo, ajustar la amplitud para bajar el volumen de salida; la forma de implementar esto con precisión es multiplicando por un factor entre 0 y 1. Lo mismo ocurre para incrementar el volumen, salvo que el factor debería ser superior a 1. También en operaciones más complejas como algoritmos de filtrado, para calcular umbrales, desfases, etcétera es imposible obviar la parte decimal de los valores.

Puesto que a lo largo de la memoria se harán referencias al punto fijo, cabe mencionar que habitualmente se utiliza la nomenclatura "QN.M" para indicar el número de bits que representan la parte entera (N) y decimal (M); por ejemplo, Q10.5 significa que 10 bits son para la parte entera y 5 para la decimal.

Suma y resta en punto fijo

Con valores en punto fijo, las operaciones de suma y resta se implementan igual que en la aritmética entera en C2. Sin embargo existe una restricción: los operandos deben utilizar el mismo número de bits para representar la parte decimal.

Si dicha restricción no se satisface, es necesario reescalar los valores para alinear las comas. Esto se realiza mediante desplazamientos, aunque puede provocar que los operandos

resulten desbordados. Otro inconveniente que surge al desplazar es la pérdida de resolución por la reducción del número de bits empleados para representar la parte decimal; existen dos formas de tratarlo: truncar (descartar bits menos significativos) o redondear (aproximar al valor más cercano).

Multiplicación y división en punto fijo

Para este tipo de operación no es necesario que estén alineadas las comas decimales, es posible utilizar directamente aritmética entera. No obstante, hace falta ajustar la escala del resultado, puesto que el número de bits que emplea este para representar la parte decimal, equivale a la suma del número de bits decimales de ambos operandos. Por ejemplo, al multiplicar dos valores uno en Q6.2 y otro en Q7.3, el resultado vendrá expresado en Q13.5.

Al reescalar puede producirse desbordamiento o pérdida de resolución, que se administran con las mismas técnicas que las descritas para la suma.

3.1.3. Audio digital: Desbordamiento

Mezclar dos señales en formato digital se reduce a una simple operación de suma de sus muestras. Esto puede suponer un problema si el resultado de la operación es tan grande que no es posible representarlo con el número de bits utilizado.

Este fenómeno se conoce como desbordamiento y hay distintas formas de mitigarlo, por ejemplo devolver el máximo valor representable (como se detalla en la sección contigua).

El inconveniente es que el sonido resulta distorsionado; sin embargo puede ser también una ventaja, puesto que muchos efectos se basan en saturar intencionadamente las muestras.

La solución tomada en el sistema para impedir el desbordamiento descontrolado se ha basado en extender el número de bits de los operadores. En la salida del sistema se reajusta a la anchura exigida por el conversor digital-analógico.

Cabe mencionar que la mayor parte de mezcladores no suman las entradas de audio sin multiplicarlas primero por un factor de reducción. El objetivo que persiguen es que el volumen sonoro del resultado no exceda el de las muestras originales, lo cual difiere

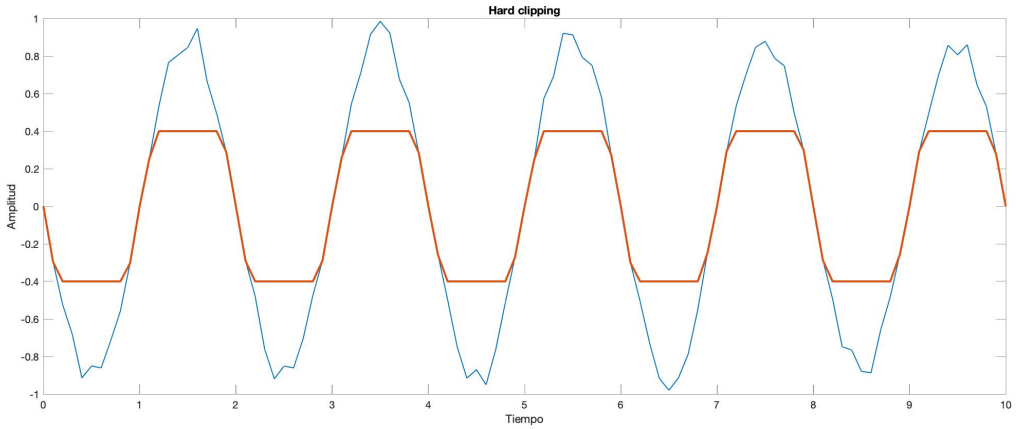


Figura 3.1: En azul la señal original, en rojo sometida a *hard clipping*.

completamente con el propósito de este *looper*, pues cuando dos instrumentos musicales suenan simultáneamente su volumen no se ve reducido.

3.1.4. Audio digital: Limitación

La limitación, también conocida como *clipping* consiste en impedir que una onda exceda un umbral. Se utiliza para impedir que el desbordamiento derive en artefactos en el sonido, así como para provocar intencionadamente efectos de sonido. [1, 5]

Hard clipping

Existen distintas variantes, la más simple (*hard clipping*) se basa en mantener el valor original cuando este no supere el umbral, y en caso contrario reemplazarlo por dicho límite. Expresado matemáticamente:

$$hardClipping(x) = \begin{cases} t, & x \leq -t \\ x, & -t \leq x \leq t \\ t, & x \geq t \end{cases}$$

Donde t es el valor umbral y x la muestra.

La figura 3.1 ilustra el efecto que el *hard clipping* produce en una onda.

Además de en los mezcladores, se utiliza en el efecto *fuzz*.

Soft clipping

Una alternativa es el *soft clipping*, que en lugar de utilizar un valor constante cuando se supera el umbral, trata de suavizar la forma de onda y evitar el "escalón" que se produce en el caso anterior.

Hay diversas formas de implementarlo, algunas de ellas requieren un procesamiento no lineal ya que es necesario conocer si la amplitud de onda va a exceder el umbral en un futuro, para comenzar a reducir su valor antes de que llegue a saturar. De este modo se lograría dicha suavidad.

Sin embargo no es necesario realizar procesado no lineal (que además induciría un retardo en la onda resultante), puesto que existen algoritmos más simples que logran efectos similares:

$$softClipping(x) = \begin{cases} -\frac{2}{3}, & x \leq -1 \\ x - \frac{x^3}{3}, & -1 \leq x \leq 1 \\ \frac{2}{3}, & x \geq 1 \end{cases}$$

Siendo x la muestra a procesar.

Sin embargo, dado que en este proyecto la implementación va a realizarse en *hardware*, se ha optado por otra aproximación que no necesita calcular el cubo (y así no tener que asumir el coste de dicha operación):

$$softClipping(x) = \begin{cases} \frac{x-t}{4} + t, & x \geq t \\ x, & t \leq x \leq t \\ \frac{x-t}{4} + t, & x \leq -t \end{cases}$$

Donde x la muestra a procesar y t el valor umbral.

Una de las ventajas de la fórmula anterior es que la división entre cuatro es inmediata en *hardware* y puede implementarse desplazando dos bits la muestra. En la gráfica 3.2 se muestra cómo el *soft clipping* actúa sobre la onda.

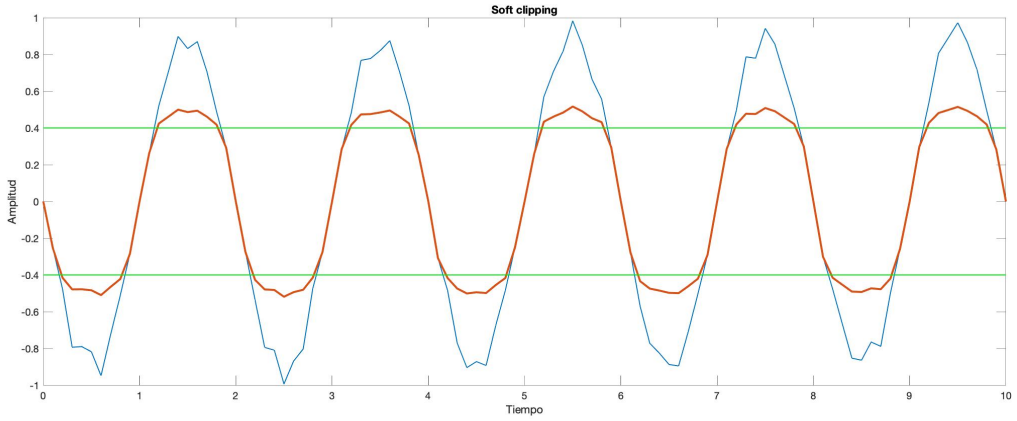


Figura 3.2: En azul la señal original, en rojo sometida a *soft clipping*. El umbral está marcado en verde.

Compresión de rango dinámico

Otra opción es la compresión de rango dinámico. La particularidad respecto a los anteriores es que afecta a toda la onda, no solo a la parte que supera el umbral. Consiste en reducir el volumen sonoro de la misma; multiplicándola por un factor entre cero y uno que varía en función de su amplitud. No produce distorsión en el sonido, pero reduce la diferencia de amplitud entre los sonidos más tenues y los de mayor intensidad.

Más allá del umbral, en este ámbito cobran importancia detalles como la relación de compresión (cuánto se reduce la amplitud respecto a la original), ataque (tiempo transcurrido hasta aplicarse la compresión, una vez superado el umbral) o tiempo de decaimiento (cuánto tarda en dejar de comprimir).

Cuando se extrema el valor de los parámetros, estableciendo un tiempo de ataque mínimo y relación de compresión muy alta, el comportamiento de la compresión se asemeja a la limitación.

En la figura 3.3 se ilustra el efecto que tiene la compresión en una señal.

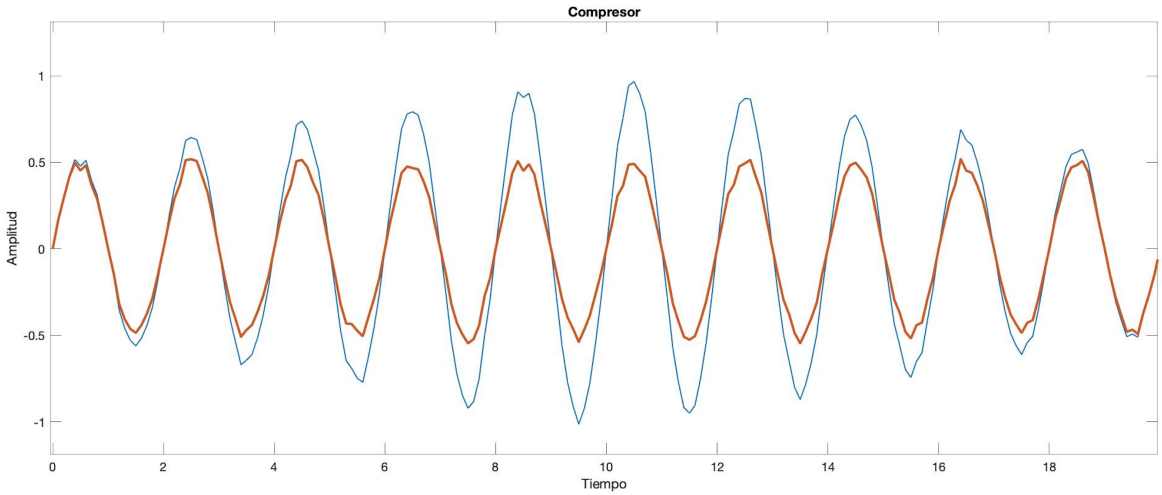


Figura 3.3: En azul la señal original, en rojo sometida a compresión.

3.1.5. Efectos de sonido basados en limitación

Overdrive

El *soft clipping* es el fundamento del *overdrive*, efecto que imita lo que ocurre al saturar un amplificador, cuando los transistores o válvulas de vacío no son capaces de incrementar más el volumen. Por ello se implementa como un limitador (*soft*), impidiendo que el valor de la señal exceda un umbral. El resultado es una distorsión suave, propia del *Rock and Roll* clásico.

Fuzz

El efecto *fuzz* está principalmente basado en *hard clipping*. No obstante, en este caso la limitación es peculiar dado su carácter asimétrico: las muestras positivas tienen un valor umbral distinto a las negativas. Para forzar el sonido a que exceda el umbral, es previamente amplificado. [1]

Al igual que el *overdrive*, distorsiona el sonido, aunque en mayor medida. También provoca la aparición de armónicos, es decir, múltiplos de la frecuencia original que enriquecen el sonido.

En la figura 3.4 puede observarse un diagrama de bloques del *fuzz*.

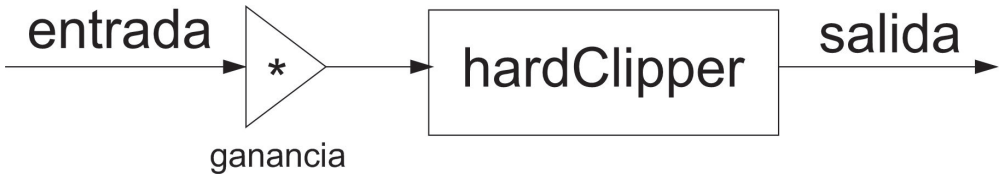


Figura 3.4: Diagrama de bloques del *fuzz*.

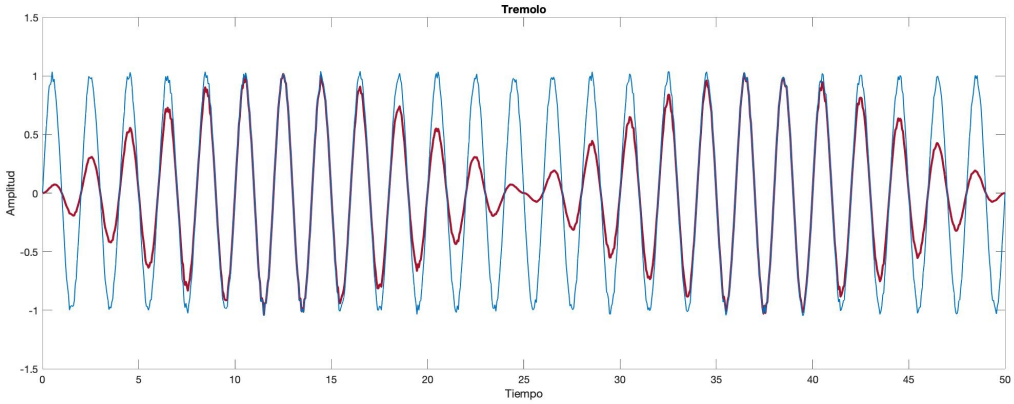


Figura 3.5: Efecto trémolo, en azul la señal original, en rojo con el efecto aplicado.

3.1.6. Efectos de sonido basados en amplitud de onda

Trémolo

El trémolo es un efecto de sonido que altera la amplitud de onda de forma regular y periódica. Internamente utiliza lo que en términos de audio se conoce como LFO (del inglés *Low frequency oscilator*), que permite generar un factor el cual oscila relativamente despacio (con frecuencia inferior a 20 Hz). Este se utiliza para modular la amplitud de la onda original.

La imagen 3.5 representa el efecto aplicado a una señal.

Un diagrama de bloques para este efecto está presente en la figura 3.6.

3.1.7. Efectos de sonido basados en retardos

Eco (*delay*)

El efecto de eco, ampliamente conocido como *delay* se basa en añadir a la señal muestras atenuadas correspondientes a un instante de tiempo anterior.

Se observan en el diagrama de bloques (figura 3.7) los factores que determinan el com-

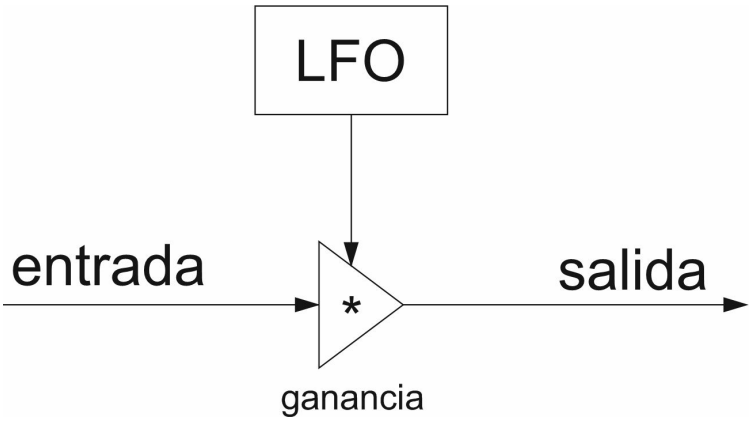


Figura 3.6: Diagrama de bloques del trémolo.

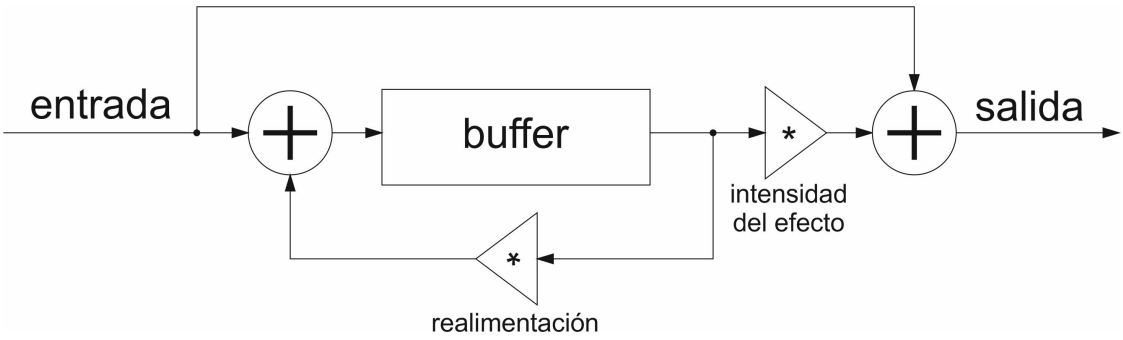


Figura 3.7: Diagrama de bloques para efectos basados en *delay*.

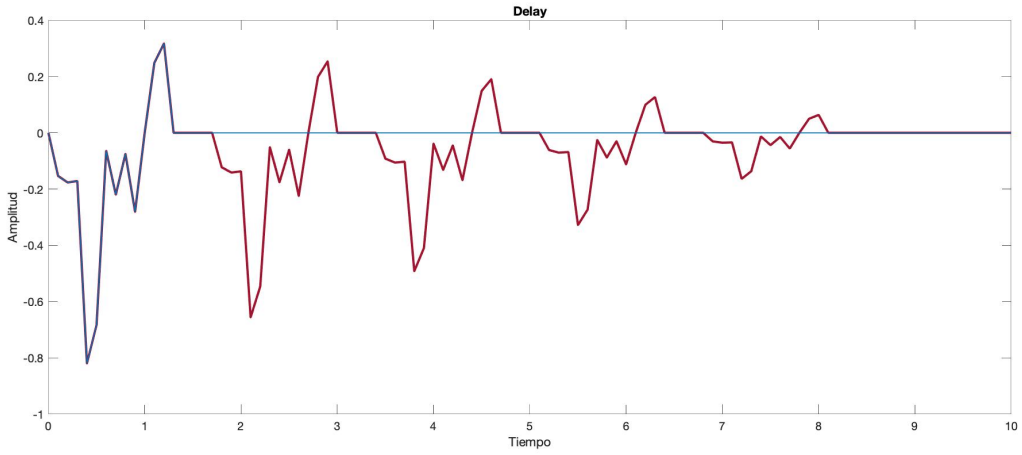


Figura 3.8: Efecto de *delay*, en azul la señal original, en rojo con el efecto aplicado.

portamiento de este efecto. En primer lugar, el tamaño del *buffer* condiciona el tiempo de retardo. También es relevante la ganancia de retroalimentación, que debe tomar un valor entre 0 y 1 para garantizar la atenuación progresiva de las señales retardadas. Por último la intensidad del efecto (es decir, la "presencia" que tendrá en el sonido resultante) depende de la ganancia aplicada a la señal proveniente del *buffer*. [1]

En la figura 3.8 se muestra el efecto aplicado a una señal.

Efecto Doppler

Las ondas de sonido se propagan a través de medios elásticos, por el aire (a 15 grados centígrados) su velocidad es de 340 metros por segundo. Sin embargo, cuando una fuente sonora se encuentra en movimiento, la velocidad relativa de las ondas respecto a un observador es distinta. Esto se traduce en que el sonido se percibe con una frecuencia distinta, su tono varía.

En general, la relación entre la frecuencia emitida y observada es la que sigue:

$$f_{observada} = \left(\frac{c \pm v_o}{c \pm v_f} \right) * f_{emitida}$$

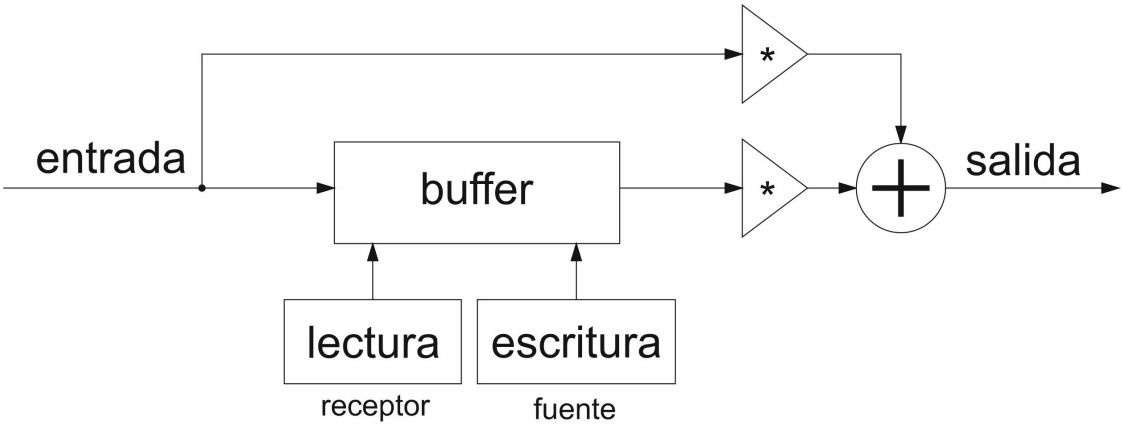


Figura 3.9: Diagrama de bloques del efecto *Doppler*.

Donde c es la velocidad de propagación en el medio, v_o la velocidad del observador y v_f la velocidad de la fuente.

Este fenómeno se conoce como efecto Doppler, y puede recrearse digitalmente alterando la velocidad de reproducción de las muestras grabadas. Un diagrama de bloques del mismo se encuentra en la figura 3.9. [5]

Flanger

El *flanger* es un efecto de sonido basado en aplicar un *delay* y mezclarlo con la señal original, mientras varía el tiempo de retardo.

Como se observa en la figura 3.10, se rige bajo un diagrama de bloques similar al presentado anteriormente para el eco, con la particularidad de que el tamaño del *buffer* está regulado por un oscilador de baja frecuencia (entre 0,1 y 1 Hz). Además no debe inducir más de 2 milisegundos de retardo. [1]

3.1.8. Efectos de sonido basados en frecuencia

Filtros FIR

Los filtros FIR (*Finite Impulse Response*) se caracterizan porque no tienen realimentación, al contrario que los filtros IIR (*Infinite Impulse Response*). Esto significa que la salida no retorna a la entrada del filtro. Su propia naturaleza hace que sean estables, por ello son

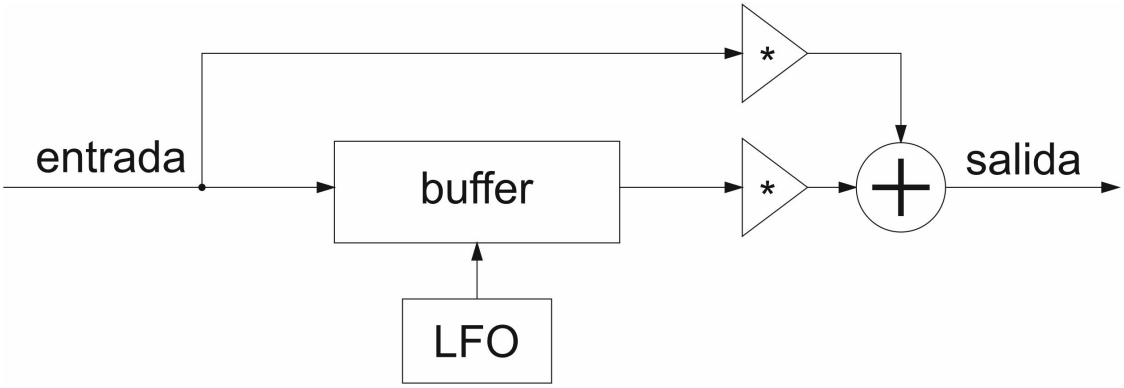


Figura 3.10: Diagrama de bloques del *flanger*.

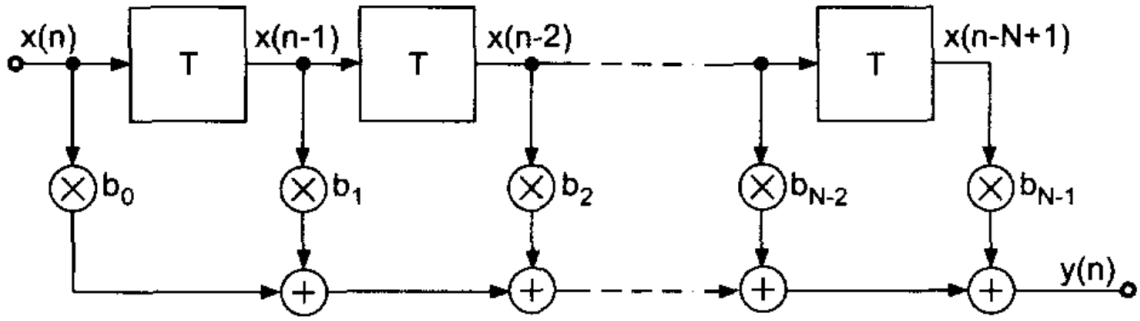


Figura 3.11: Fuente: *DAFX: Digital Audio Effects*. [1]

comúnmente empleados en el ámbito del audio digital. [6, 7]

Su estructura atiende al diagrama de bloques de la figura 3.11 (x es la entrada, y la salida).

Se observa que las muestras atraviesan secuencialmente cada etapa, tal que cada una de ellas es multiplicada por un coeficiente (o peso). La suma de todos es el valor final.

El valor de una muestra filtrada puede representarse con la siguiente función:

$$salida = \sum_{i=0}^{N} b_i * x[n-i]$$

Siendo N el número de etapas del filtro, x[n] la señal de entrada y b los coeficientes.

Emplear un número de etapas elevado en en filtro permite reducir el error del mismo, haciendo más preciso el corte de las frecuencias. Sin embargo, este tipo de filtro induce un retardo, que también depende directamente del número de etapas. Está definido por la siguiente fórmula.

$$d = \frac{N-1}{2}$$

Un detalle importante es también la paridad del número de etapas; en caso contrario debe tenerse en cuenta que la respuesta en frecuencia es cero para ciertos puntos, algo inapropiado para filtros paso alta o paso baja.

Tampoco es trivial la selección de los coeficientes, a grandes rasgos se distingue entre simétricos y asimétricos. Estos últimos añaden un desfase de noventa grados a la señal, lo cual implica ciertas restricciones.

Por los motivos expuestos en los párrafos anteriores en este proyecto se ha optado por utilizar un número impar de etapas y coeficientes simétricos. Además se han utilizado constantes precalculadas, ya que las frecuencias de muestreo y corte son fijas; de este modo, el coste computacional es únicamente el de la convolución.

Ventana de Hamming

Las ventanas son funciones matemáticas simétricas entorno al punto medio de un cierto intervalo, fuera del cual su valor corta con cero.

En este caso particular se ha utilizado la ventana de Hamming como función para obtener los coeficientes del filtro. La decisión de emplear esta se debe a la respuesta suave que proporciona en el filtro, en comparación con otras funciones que generan "rebotes" en la señal al aproximarse a la frecuencia de corte. Otra ventaja es su carácter simétrico, que permite ahorrar memoria a la hora de almacenar las constantes.

La ventana de Hamming se define:

$$w(n) = w(-n) = 0,54 - 0,46 * cos\left[\frac{\pi * (M - n)}{M}\right]$$

Siendo M:

$$M = \frac{N - 1}{2}$$

Para obtener los coeficientes del filtro FIR se deben multiplicar los valores de obtenidos mediante la ventana de Hamming por unas constantes calculadas para las características del filtro.

Para el paso baja se utiliza la siguiente fórmula:

$$hb(n) = \frac{-sin((2pi * w_L/Fs) * (M + 1 - n))}{(M + 1 - n) * \pi}$$

Para el paso alta:

$$ha(n) = \frac{sin((2pi * w_H/Fs) * (M + 1 - n))}{(M + 1 - n) * \pi}$$

Siendo Fs la frecuencia de muestreo, y w_L y w_H las frecuencias de corte del paso baja y paso alta respectivamente.

Finalmente los coeficientes a utilizar en el filtro FIR se obtienen:

$$coef_{baja}(n) = w(n) * hb(n)$$

$$coef_{alta}(n) = w(n) * ha(n)$$

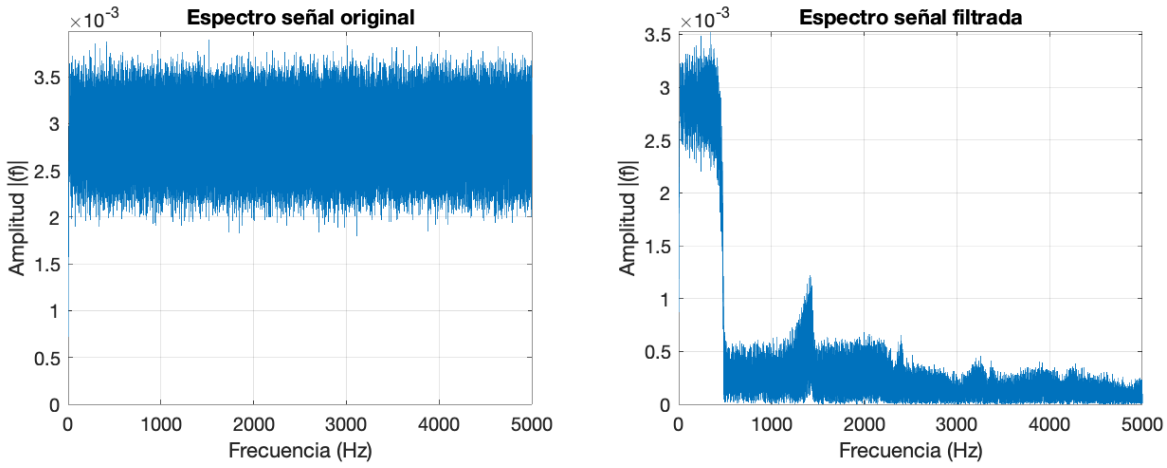


Figura 3.12: Espectro de frecuencias de un fragmento de audio que barre desde 0 hasta 5000 Hz. A la izquierda la original, a la derecha la sometida a un filtro paso baja con aritmética en punto fijo (frecuencia de corte: 440 Hz).

Un detalle sobre estos coeficientes es que son números reales, y será necesario trabajar con ellos en punto fijo. Esto conlleva una pérdida de fidelidad en el sonido, puesto que la precisión es limitada y se produce un cierto error de cuantificación; sin embargo, con 15 bits para la parte decimal y 17 para la entera (Q17.15) se han obtenido resultados más que aceptables. Se observa en la figura 3.12 el espectro de frecuencias de una señal de audio que crece desde 0 a 5000 Hz, pasada por un filtro paso baja cuya frecuencia de corte se ha fijado en 440 Hz. Dicho filtro ha sido implementado en *Matlab*.

El error se debe por un lado al algoritmo, aunque también a la cuantización que implica utilizar aritmética en punto fijo. Para el segundo caso puede observarse en la figura 3.13 como el error de cuantización desaparece al utilizar aritmética en punto flotante. En cualquier caso, es imperceptible para el oído humano.

3.2. Placa de prototipado (*Nexys 4 DDR*)

La herramienta principal y fundamental para la realización de este proyecto ha sido una placa de prototipado con FPGA integrada; el modelo en cuestión es la *Nexys 4 DDR* (véase

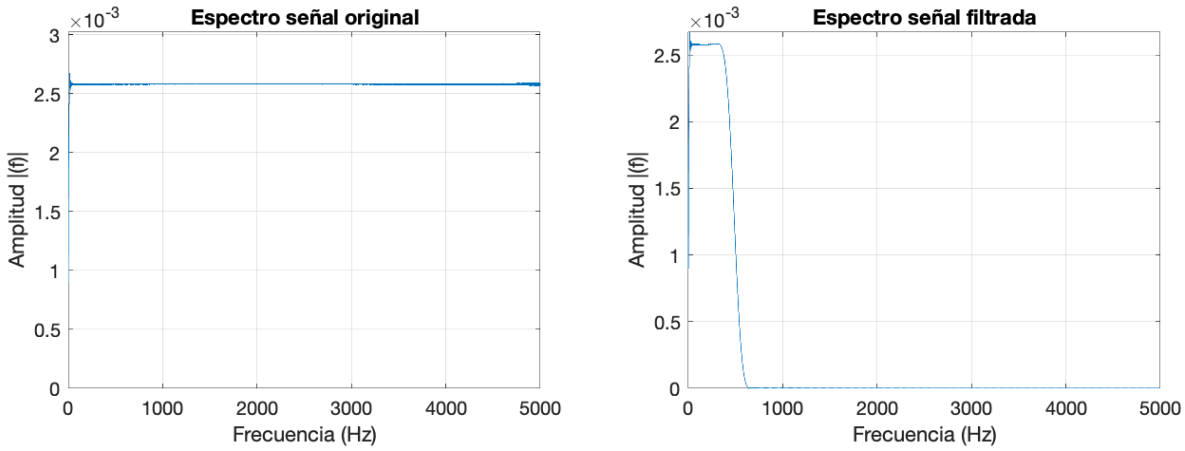


Figura 3.13: Espectro de frecuencias de un fragmento de audio que barre desde 0 hasta 5000 Hz. A la izquierda la original, a la derecha la sometida a un filtro paso baja con aritmética en punto flotante (frecuencia de corte: 440 Hz).

figura 3.14).

3.2.1. FPGA

Las FPGAs son dispositivos programables, capaces de reconfigurar su interconexionado interno para recrear circuitos de electrónica digital. Permiten implementar arquitecturas *hardware* fácilmente; así como probar infinidad de prototipos, puesto que el proceso de configuración es reversible.

El modelo utilizado ha sido una *Xilinx Artix-7 (XC7A100T-1CSG324C)*, que cuenta con 15850 *slices*: celdas con ocho *flip-flops* (que son memorias estáticas de acceso aleatorio, con un bit de anchura) y cuatro LUTs. Estos últimos constituyen tablas de verdad y permiten implementar cualquier función lógica de seis entradas.

Además proporciona 240 DSP, módulos para procesamiento digital de señal que contienen dos sumadores, un multiplicador de 25 por 18 bits y un acumulador de 48 bits. Estos resultan imprescindibles para acelerar los cálculos aritméticos.

Otra característica remarcable de la FPGA son los 4860 KiB de *block RAM*, un tipo de memoria integrada cuyo tiempo de acceso es de un ciclo de reloj. En este caso se divide en

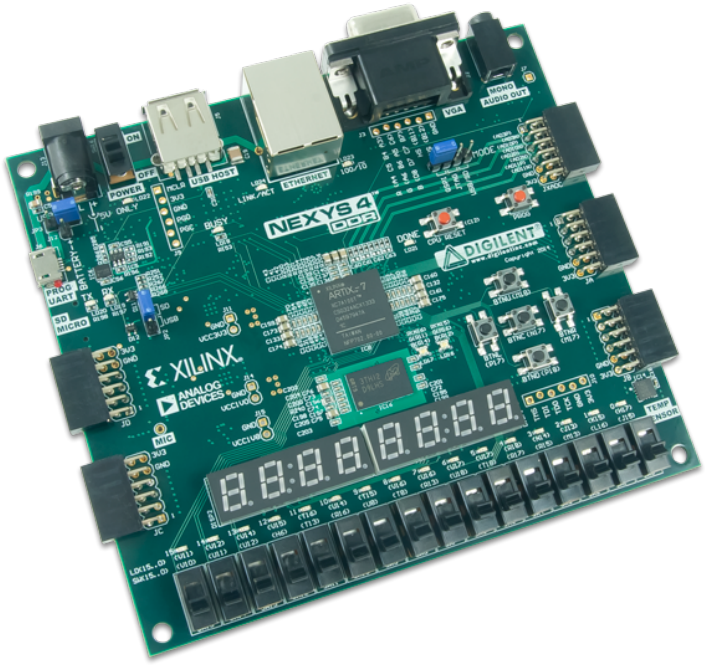


Figura 3.14: Fuente: reference.digilentinc.com.

bloques de 36 KiB. También es posible utilizar hasta un 25 % de los LUTS como memoria. [8]

3.2.2. Memoria DDR2 DRAM

En la *Nexys 4 DDR* hay también una SDRAM *off-chip*. Este tipo de memoria es "dinámica de acceso aleatorio" por la necesidad de refrescar periódicamente la información de sus celdas; ya que almacenan la información con condensadores, los cuales se descargan relativamente rápido e incorpora un interfaz síncrono. A pesar de ser más lenta que la *block RAM*, es más barata. Por ello también es posible disponer de mayor capacidad en estas memorias, la que hay instalada en la placa presenta 128 MiB de capacidad. Se trata de un módulo *ISSI IS43/46DR16640C*.

La interfaz que utiliza es DDR2, que es un estándar surgido a principios de la década de los 2000. Implementa la tecnología *dual rate*, por lo que es capaz de proporcionar datos en ambos flancos (subida y bajada) del reloj; asimismo puede operar internamente al doble

de la frecuencia que el bus de datos, para reducir la latencia o incrementar el ancho de banda. Aunque admite diversas configuraciones, funciona con tiempos de ciclo de hasta 2,5 nanosegundos. Respecto a su estructura, tiene ocho bancos con 16 millones de palabras cada uno. [9]

Sensor de temperatura y conversor analógico digital

El sensor de temperatura integrado en la FPGA es necesario para el correcto funcionamiento de la memoria DDR, en concreto para alinear determinadas señales (DQ y DQS).

Dicho sensor está conectado a un conversor analógico digital (XADC) que hace posible tomar muestras periódicamente del valor de la temperatura actual. También se encuentra incorporado en la FPGA.

Ambos forman deben formar parte del controlador de memoria a desarrollar, tal que sea este el que administra las correcciones necesarias para alinear satisfactoriamente las señales DQ y DQS.

3.2.3. VGA

VGA es un estándar utilizado en controladores gráficos para transmitir los datos de vídeo a un monitor (figura 3.15).

El VGA utilizado en el proyecto dispone de 12 bits de profundidad de color, cuatro por cada componente (rojo, verde y azul). La resolución es de 640 píxeles de ancho por 480 de alto, la frecuencia de refresco son 60 Hz.

Los monitores son esencialmente matrices de puntos, por ello para la transmisión de cada *frame* se emplea un protocolo serie, tal que se envía secuencialmente para cada píxel el color asociado. Es decir, se realiza un "barrido" de todas las filas y columnas. Sin embargo VGA es asíncrono, no se transmite el reloj, ambos extremos deben comunicarse a 25 MHz. Para sincronizar las tramas de bits entre emisor y receptor utiliza dos señales: *hsync* y *vsync*. La primera indica que todos los píxeles de una fila han sido enviados y la segunda que todas las filas de la matriz han sido enviadas.

Figura 3.15: Conector VGA.

Al iniciarse la transmisión de un *frame*, *hsync* y *vsync* realizan una transición de baja a alta, y transcurridos ocho ciclos se transmiten secuencialmente los 640 píxeles a la velocidad pautada por el reloj de 25 MHz. Otros ocho ciclos después de completarse la transmisión, *hsync* vuelve a baja durante 96 ciclos. Este proceso se repite para las 480 líneas de la pantalla. Finalmente, cuando todos los píxeles hayan sido enviados el emisor *vsync* pasa de alta a baja, manteniéndose en este estado otro cierto tiempo. El proceso se repite continuamente.

Como se observa, hay una serie de ciclos de espera (en los que no se envía información) después de enviar cada línea. De este modo para los 640 píxeles de cada línea se emplean 800 ciclos (resultando una frecuencia de 31,250 KHz para el envío de las líneas). Y para las 480 líneas visibles, se añaden otras 45 sin transmitir píxeles (que finalmente hace que los fotogramas se redibujen a 59,52 Hz).

Estos tiempos de espera se deben a los monitores de rayos catódicos, que mediante un haz de electrones plasmaban el color en una superficie de fósforo situada tras el cristal. La cuestión es que la emisión de electrones debía detenerse mientras el haz volvía a la posición de origen (para pintar la nueva línea), de lo contrario se plasmaría en la pantalla un "hilo" dibujado por el paso de los electrones [10].

Para visualizar las imágenes se ha utilizado un monitor externo de 19,5 pulgadas (*Acer*

V206HQL).

3.2.4. PS/2 a través de USB

La placa también incluye un controlador compatible con USB HID (parte del estándar USB que engloba los dispositivos de interfaz humana). De este modo, a través del puerto USB que hay en la *Nexyx 4 DDR* es posible conectar teclados y ratones [11].

La comunicación se realiza mediante el protocolo PS/2. Este es serie y síncrono, por tanto utiliza una línea para datos y otra para el reloj.

Las tramas contienen ocho bits de datos y tres de control: *start*, *stop* y paridad. Cuando en el teclado tienen lugar eventos relacionados con alguna tecla, se envían una o varias tramas al receptor, con un código para identificar a la misma (3.16). Para las pulsaciones se distingue entre teclas corrientes y especiales, en caso de estas últimas se envían dos tramas (primero una con el código "E0" en hexadecimal, y después otra asociada a la tecla en cuestión). Esto es necesario porque ocho bits no son suficientes para codificar todas las teclas, teniendo en cuenta las extensiones que se han realizado posteriormente en el estándar (funciones multimedia, controles adicionales...). Para comunicar cuando una tecla ha sido despulsada, se envía una trama con el código "F0" y a continuación el mensaje enviado en la pulsación. En el caso de las teclas especiales son tres tramas "E0", "F0" y la correspondiente a dicha tecla. [12]

3.2.5. Memoria flash

Para almacenar el fichero de configuración con el diseño a cargar en la FPGA, es posible utilizar la memoria *flash* no volátil que hay en la placa. De este modo es posible operar con la placa sin necesidad de un ordenador para programar la FPGA.

Para habilitar esta función en la placa es necesario almacenar el diseño en la memoria (se realiza mediante el mismo *software* utilizado para programar la FPGA) y configurar un *jumper* para cargar la configuración en el arranque.

El modelo en cuestión es una *Spansion S25FL128S*. Es de tipo NOR, cuenta con 16

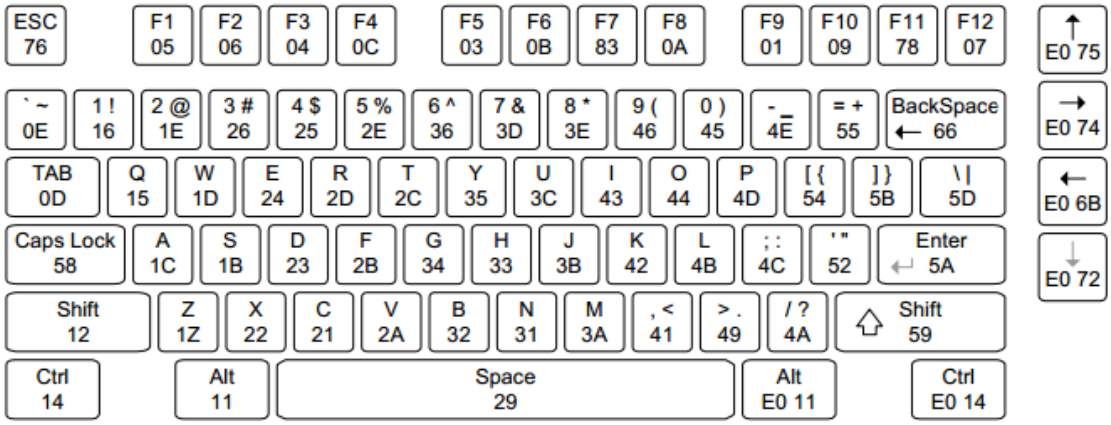


Figura 3.16: Fuente: reference.digilentinc.com.

MiB de memoria y es capaz de realizar transferencias a 66 MiB por segundo en modo DDR. Aunque por motivos de compatibilidad con la placa se ha trabajado a 33 MiB por segundo. Puede llegar a consumir 61 mA al funcionar en modo de lectura. En cualquier caso el rendimiento de esta memoria no es crítico, puesto que solo será utilizada puntualmente.

El protocolo de intercambio de datos con esta memoria es QSPI, una extensión del BUS SPI que utiliza cuatro líneas para la comunicación (en lugar de dos).

3.2.6. Otros componentes

Además de los ya mencionados, la *Nexys 4 DDR* integra otros componentes tales como *switches*, pulsadores, leds, micrófono, puerto *Ethernet*, lector de *microSD*, acelerómetro, salida de audio y un *display* 7 segmentos. Sin embargo estos no se utilizan en el proyecto, aunque algunos de ellos (principalmente leds, *switches* y pulsadores) si han tenido cabida en tareas de depuración para visualizar o controlar ciertos aspectos del sistema.

3.2.7. PMODs

Además de los componentes que integra la placa de prototipado es posible conectar a la misma diversos periféricos a través de los cuatro puertos PMOD que presenta. Este estándar físico abierto, desarrollado por el mismo fabricante de la placa (es decir, *Digilent*) propor-

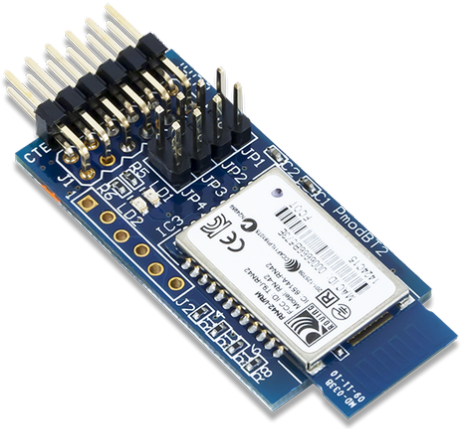
ciona por cada puerto ocho líneas de entrada/salida en serie de datos, dos de alimentación (3,3 voltios) y dos de tierra.

Hay multitud de módulos compatibles con este estándar: SPI, I2C, pantallas LCD... En este proyectos se han utilizado dos, descritos a continuación.

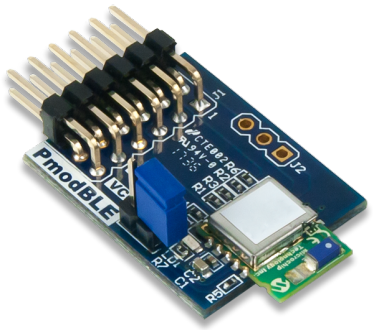
3.2.8. *Bluetooth*

Bluetooth es un estándar de comunicación sin cables, orientado a dispositivos móviles.

Durante el transcurso del proyecto se han empleado dos módulos PMOD para lograr establecer comunicaciones a través de *Bluetooth*. En primer lugar se utilizó un *Digilent PmodBT2*, compatible con el estándar *Bluetooth* 2.1; sin embargo, fue posteriormente reemplazado por un *Digilent PmodBLE*. Este soporta el estándar*Bluetooth* 4.2, más reciente y optimizado en lo que al consumo de energía se refiere (véase figura 3.17). [13–15]



(a) *Digilent PmodBT2*.



(b) *Digilent PmodBLE*.

Figura 3.17: Fuente: reference.digilentinc.com.

La ventaja de utilizar estos PMOD es la transparencia casi absoluta de los detalles de la comunicación, pues disponen de un microcontrolador integrado que se encarga de resolver las capas más bajas del protocolo, desde el establecimiento de la conexión hasta el control de errores. Tanto los datos recibidos como los datos a enviar se intercambian entre el periférico

y la FPGA mediante el estándar RS-232.

Este último es un protocolo serie asíncrono (puesto que no se transmite la señal de reloj), que utiliza dos líneas, RX y TX, para la recepción y transmisión respectivamente. Las transferencias se realizan a una velocidad prefijada.

En concreto, el chip que integra el PMOD utilizado finalmente en el diseño es un *Microchip RN4871*. Incorpora internamente una antena cerámica, tal que no es necesaria una externa. Esta opera entre 2,402 y 2,480 GHz (canales del 0 al 40), su rango de alcance llega a los 10 metros de distancia. En cuanto a energía, mientras está conectado e intercambiando mensajes consume entorno a 3 mA. Además permite comunicaciones cifradas (AES128).

Es posible configurarlo remotamente una vez establecida la comunicación con él, mediante un modo de funcionamiento especial que recibe comandos en texto plano. Para ello implementa un servicio *Transparent UART*.

Bluetooth* en la aplicación *software

Al otro lado de la comunicación, en la aplicación *software* es necesario vincular el controlador *Bluetooth* con el PMOD. Para ello es necesario especificar tres identificadores únicos de 128 bits, uno para establecer la conexión, otro para el envío y el último para la recepción de datos.

El resto de detalles de la comunicación quedan resueltos por las bibliotecas del sistema operativo.

3.2.9. I2S

I2S (*Integrated Interchip Sound*) es un bus serie síncrono diseñado para la transmisión de audio estéreo en formato PCM. Este último permite representar las ondas de sonido (u otro tipo) como conjuntos de muestras, tal que cada una de ellas indique (en C2) la amplitud de la señal en el instante de tiempo al que corresponden. Por tanto, la fidelidad del audio digitalmente codificado depende principalmente de la anchura de las muestras y la frecuencia de muestreo; cuanto mayores sean ambos, más precisa será la representación.

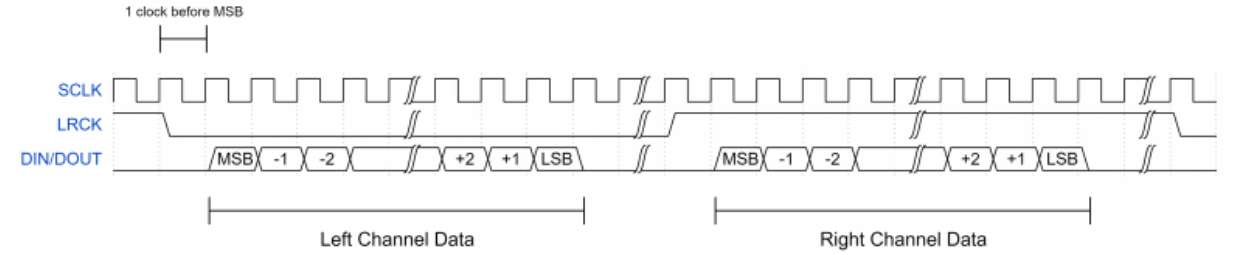


Figura 3.18: Fuente: reference.digilentinc.com.

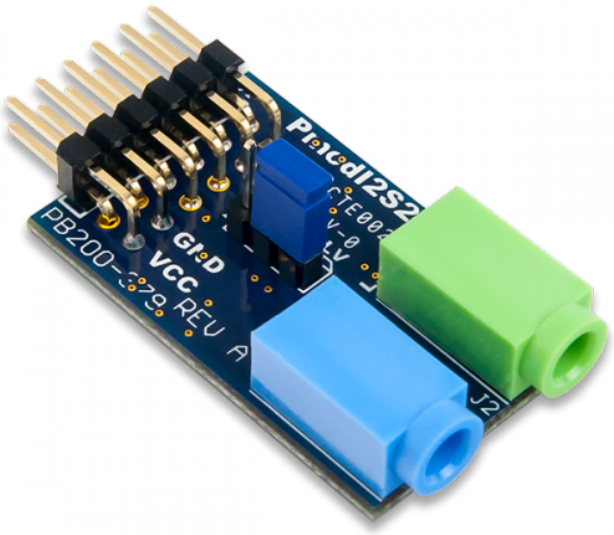


Figura 3.19: Fuente: reference.digilentinc.com.

El protocolo utiliza tres líneas para señales de reloj: MCLK (reloj principal), SCLK (para la transmisión bit a bit de las muestras) y LRCLK (para indicar canal izquierdo/derecho). Para los datos utiliza una línea por cada canal estéreo [16].

La frecuencia de LRCLK debe ser equivalente a la frecuencia de muestreo, y tanto SCLK como MCLK deben guardar una relación exacta con ella (depende del códec, las admitidas se indican en las especificaciones).

En el caso particular del PMOD empleado en el proyecto (véase figura 3.19), este presenta dos códecs de audio independientes, uno para la entrada (conversor analógico-digital) y otro para la salida (conversor digital-analógico).

El dispositivo de entrada es un *Cirrus Logic CS5343/4*, tiene una resolución de 24 bits y es capaz de muestrear a 96 KHz. En la salida se trata de un *Cirrus Logic CS4344/5/8*, también con 24 bits de resolución, aunque la frecuencia de muestreo es superior: 192 KHz. Cada uno de ellos utiliza sus propias líneas de reloj [17, 18].

3.3. *IP soft cores*

Aunque los *IP cores* no constituyen ningún componente material, siguen siendo módulos *hardware*, se trata de diseños de circuitos cargables en la FPGA. Sus siglas IP provienen de *Intelectual Property* ya que son típicamente proporcionados por los fabricantes, que se encargan de diseñarlos, verificarlos y distribuirlos.

En muchos escenarios, estos pueden facilitar bastante determinadas tareas; en este proyecto se ha recurrido a los siguientes (todos ellos desarrollados por la empresa *Xilinx*).

3.3.1. *Memory Interface Generator*

Este IP genera interfaces de acceso a memoria DRAM de tipo DDR3 o DDR2, para ello es preciso configurarlo utilizando los parámetros apropiados para el módulo físico a utilizar. Entre ellos la frecuencia de reloj, la relación respecto al reloj del controlador, el número de bancos, la anchura de palabra, la disposición de los pines de la memoria, etcétera [19].

3.3.2. *Clock Generator*

El *Clock Generator* sirve para generar hasta ocho relojes derivados a partir de una frecuencia base prefijada. Para los casos en que no es posible obtener la deseada, produce la aproximación más cercana posible [20].

3.3.3. *Integrated Logic Analizer*

Este módulo juega un papel fundamental en la depuración de los diseños implementados, una vez cargados en la FPGA. Permite monitorizar señales en tiempo real, siendo posible

configurar disparadores para programar capturas de datos; tal que cuando se reúnan una serie de condiciones se empiece a guardar el comportamiento de las señales por un tiempo prefijado.

No obstante presenta alguna limitación, puesto que funciona ligado a una frecuencia de reloj y por ello no es posible analizar al completo el comportamiento de las señales (por ejemplo *glitches* o rebotes). Por otro lado, dada la naturaleza del *hardware* (por su alta velocidad y el funcionamiento inherentemente paralelo), para observar la evolución de las señales es estrictamente necesario "capturar" primero los cambios producidos durante un lapso de tiempo, y analizarlo después. Tampoco es posible almacenar más de 1024 de ciclos en las capturas de datos.

No obstante, al tratarse de *hardware* estos límites son muy difíciles de salvar [21].

3.4. Herramientas *software*

3.4.1. EDA y descripción *hardware*

La descripción *hardware* se ha realizado íntegramente en lenguaje VHDL (*Very High Speed Integrated Circuit Hardware Description Language*), concretamente el estándar de 1993 [22].

La herramienta EDA empleada para la síntesis e implementación del diseño es *Vivado 2018.2*. La síntesis es el proceso por el cual se obtienen diseños en términos de componentes interconectados a partir de las descripciones a nivel RTL (de transferencia de registro). Esta traducción va ligada a un complejo análisis que verifica la correctitud del diseño a distintos niveles. Además somete las descripciones a una profunda optimización con vistas (entre otros) a ahorrar recursos, área (es decir, el espacio físico que ocupa el diseño), reducir caminos lógicos y satisfacer todas las restricciones.

Vivado también ha sido la herramienta de la cual se han obtenido y donde se han configurado los *IP soft cores* utilizados, así como la empleada en las tareas de depuración y análisis del comportamiento de las señales. Además se ha utilizado para programar la FPGA

y grabar el diseño en la memoria *flash* de la placa.

Como cabría esperar, esta herramienta también ha sido empleada en la obtención de estadísticas de utilización de recursos, análisis de tiempos o consumos del diseño *hardware*.

3.4.2. Simulación de efectos de audio

Para las pruebas y simulaciones previas a la implementación de los efectos de sonido en el sistema, se ha utilizado *Matlab R2019b*, así como su componente interno *Simulink*. También han sido de utilidad para la generación de gráficos que representan el comportamiento de la forma de onda resultante de aplicar efectos al sonido.

Puntualmente para realizar ciertas pruebas con efectos basados en retardos, se han elaborado *scripts* en lenguaje *Python*.

3.4.3. Programación *software* (*iOS*)

La aplicación *software* destinada al control remoto del *looper* se ha desarrollado enteramente en lenguaje *Swift* (versión 5.1), un lenguaje compilado de propósito general.

La generación de binarios se ha realizado con el compilador LLMV. Para ello, además de las tareas de depuración se ha utilizado el entorno *Xcode 11.4*.

3.4.4. Dibujo asistido por ordenador

El diseño de una caja para el producto final ha implicado utilizar herramientas de dibujo asistido por ordenador (o CAD, por sus siglas en inglés) 3D.

Fundamentalmente se ha utilizado *Autodesk Inventor 2018*, este *software* basa su funcionamiento en operaciones paramétricas, de modo que resulta sencillo modificar las piezas incluso en estados avanzados del diseño, a costa de complicar sensiblemente el proceso de modelado.

Para las partes en con formas complejas (es decir, los logotipos e inscripciones) ha sido necesario realizar operaciones con sólidos tridimensionales en la aplicación *Autodesk Autocad*

2018. Esta es más flexible que la anterior para dicha labor, no obstante alterar ciertos aspectos de las piezas una vez diseñadas es mucho más tedioso.

Para la fabricación automática de las piezas se precisa generar ficheros *G-code* de control numérico. En ellos se detallan las instrucciones a ejecutar por cada componente de la impresora 3D (como los motores, los dispositivos de fusión del material o el calentador de la plataforma). Este proceso se ha realizado mediante el *software Cura 14*.

Finalmente, *Repetier Host* ha sido el programa empleado para la interpretación de las instrucciones y control de la impresora.

3.4.5. Gráficos vectoriales

El diseño de los gráficos presentes en esta memoria para los diagramas arquitectónicos, además de logotipos u otras figuras utilizadas en el proyecto se ha llevado a cabo con la aplicación de dibujo vectorial *Corel Draw 2019*.

Para el retoque fotográfico de algunas de las imágenes de dispositivos de audio mostradas también se ha recurrido al software *GIMP*.

3.4.6. Generación de documentación

Para la redacción de esta memoria se han utilizado los editores de texto *Vim* y *Atom*.

La generación de los documentos finales debidamente formateados se ha realizado con el sistema *LaTeX* (en conjunto con *BibTeX* para las referencias), a través de la aplicación *TeXShop*.

3.4.7. Control de versiones

El tanto el código *VHDL* del proyecto, como la aplicación *software* (en *Swift*), los diseños gráficos y los planos de piezas en 3D se han sometido por motivos de seguridad a control de versiones con la herramienta *Git*. Los repositorios han sido alojados en el servicio en línea *GitLab*.

La coordinación entre los tutores y el alumno para la corrección de la memoria se ha llevado a cabo mediante la plataforma en línea *Overleaf*, que permite la edición colaborativa de documentos de texto en formato *LaTeX*.

3.5. Herramientas *hardware*

3.5.1. Medios de desarrollo

Para la ejecución del *software* listado en la sección previa, se han utilizado dos computadores domésticos, uno de sobremesa y otro portátil. De este modo ha sido posible realizar en paralelo ciertas tareas; especialmente aquellas cuyo tiempo necesario para completarse es grande, mientras que su elevado uso de recursos en la máquina imposibilitaba trabajar con otro programa. Esta situación se ha dado principalmente al sintetizar con *Vivado*, sobre todo en las etapas más avanzadas del proyecto.

Ambos equipos están equipados con microprocesador *Intel Core i7* (el sobremesa tiene arquitectura *Sandy Bridge*, mientras que el portátil es *Skylake*) y 16 GiB de memoria principal.

3.5.2. Plataforma objetivo para la aplicación *software*

El dispositivo dónde se ejecuta la aplicación *software* es un tablet, modelo *Apple iPad Pro* de 12,9 pulgadas.

Dispone de una pantalla táctil razonablemente grande que permite incorporar mayor número de elementos en la interfaz gráfica, lo cual contribuye a facilitar el control del *looper*, que es su cometido.

Como es evidente, dada la importancia del requisito, es compatible con el estándar *Bluetooth* 4.1. Aunque el módulo de comunicación conectado a la FPGA (anteriormente descrito) está diseñado para cumplir con la versión 4.2 de la especificación, no supone ningún problema, pues es compatible con versiones anteriores.

3.5.3. Impresora 3D

La fabricación del chasis diseñado para integrar todos los componentes del *looper* se ha realizado mediante una impresora 3D de modelado por deposición fundida.

El material empleado ha sido plástico ABS, posible gracias al sistema de cama caliente que implementa la máquina. La elección del material está motivada por su compromiso entre calidad del acabado y buena resistencia a los esfuerzos de corte y tracción (los más sensibles al fabricar con esta tecnología). Sin embargo, el ABS requiere mayor delicadeza en la impresión, y por tanto más tiempo puesto que el cabezal debe moverse más despacio. Además, la necesidad de mayor temperatura puede acarrear problemas de deformación debido a las contracciones que tiende a sufrir este tipo de plástico.

Capítulo 4

Arquitectura *hardware* e implementación de *LoopMAN*

En la sección 2.3 se ha proporcionado una visión general de la arquitectura de este sistema, acorde a la figura 2.14, la cual describe brevemente los principales módulos del sistema y la forma en que se comunican.

Cabe remarcar que todo el diseño se ha elaborado teniendo muy presente la escalabilidad y la paralelización de los componentes, por ello en el resultado se aprecia cómo la arquitectura presenta ambas virtudes. Más allá del sistema final, esta creación constituye una plataforma adaptable y escalable, capaz de crecer considerablemente, integrando nuevas funcionalidades o perfeccionando las existentes; todo sobre el esqueleto original y puramente en *hardware*, con la innumerable lista de ventajas que ello conlleva.

Por ejemplo, suponiendo que se amplía la memoria, extender el número de pistas a 16, 32 o 64 sería trivial. Lo mismo sucede con los efectos o el número de entradas y salidas. Incluso podrían implementarse nuevos componentes en el flujo del sonido, tales como un ecualizador multibanda después del controlador de efectos o una interfaz MIDI en el módulo de control. La única complejidad de extender el sistema reside en implementar los nuevos componentes, la integración en lo ya existente es inmediata. Yendo más allá incluso podría construirse un sistema de audio distinto al *looper*, utilizando la misma base.

En el repositorio `https://gitlab.com/loopman-ndc/loopman-hw` puede encontrarse la

descripción VHDL completa del *LoopMAN*.

4.1. Características y parámetros del diseño

Hay ciertos aspectos del diseño que son paramétricos, es decir, que alterando algunos valores de la descripción en VHDL es posible ampliar (o reducir) horizontalmente el sistema. Todos ellos se listan en la siguiente tabla.

Aspectos cuantitativos del diseño		
Característica	Paramétrico	Valor utilizado
Número de pistas	No	8
Memoria asignada por pista	No	16 MiB
Número de entradas	Sí	4
Número de salidas	Sí	2
Número de efectos	Sí	9
Tamaño del buffer de *Delay* (efecto)	Sí	32 KiB
Tamaño del buffer de *Jamón* (efecto)	Sí	16 KiB
Tamaño del buffer de *Flanger* (efecto)	Sí	512 bytes
Instanciación individual de filtros (efectos)	Sí	Activado

Por otro lado también hay características relativas al funcionamiento del sistema que pueden ajustarse en tiempo real. A continuación se muestra otra tabla que incluye las mismas. De estas se indica si son ajustables por el usuario o es el sistema el que regula su valor automáticamente.

Parámetros de funcionamiento			
Módulo	Párametro	Ajuste tiempo real	Rango de valores
Pistas	Volumen	Sí	0-15 (resolución: 1)
	Margen de sincronía	No	0,167 s
	Umbral *autorec*	No	6,7 %
	Overdub	Sí	0-15 (resolución: 1)
	Compases del *loop*	Sí	1-32 (resolución: 1)
Master	Volumen	Sí	0-15 (resolución: 1)
	Umbral compresión	Sí	0-1,0 (resolución: 2^{-4})
	Ganancia compresión	Automático	0-1,0 (resolución: 2^{-4})
	Muestras sobre umbral	No	20 %
	Tamaño de ventana	No	5,24 ms
Delay	Retardo	Sí	0-0,168 s (resolución: 0,011 s)
	Ganancia realimentación	No	0,5

Overdrive	Umbral Ganancia	Sí Automático	0-1,0 (resolución: 2^{-4}) 0-7 (resolución: 1)
Jamón	Periodo oscilación Ganancia efecto Ganancia original	Sí No No	0-0,168 s (resolución: 0,011 s) 0,5 0,5
Fuzz	Umbral Ganancia previo Ganancia atenuador	No Sí Automático	Máx. 25 bits 0-3 (resolución: 1) 0,6-1 (resolución: 2^{-4})
Compresor	Umbral Ganancia de compresión Muestras sobre umbral Tamaño de ventana	Sí Automático No No	0-1,0 (resolución: 2^{-4}) 0-1,0 (resolución: 2^{-4}) 10 % 2,62 ms
Trémolo	Periodo oscilación Ganancia	Sí Automático	0,08-1,34 s (resolución: 0,08 s) 0,2-1 (resolución: 2^{-4})
Flanger	Periodo oscilación Ganancia	Sí Automático	0-2,62 ms (resolución: 0,17 ms) 0,2-1 (resolución: 2^{-4})
Filtro PB	Frecuencia de corte	Sí	15K, 3,5K, 1K, 400, 220 Hz
Filtro PA	Frecuencia de corte	Sí	25, 40, 80, 140, 190 Hz

4.2. Comunicación entre módulos

En lo que respecta al intercambio de información entre módulos, puede realizarse una distinción en tres tipos de datos: audio, señales de control y señales de estado.

El audio pasa de un módulo a otro, tal y como se menciona en el apartado 2.3: *Arquitectura (visión general)*. Para ello, una vez procesadas las muestras, los módulos la almacenan en un registro a su salida y notifican el fin de procesamiento mediante un pulso. En el diseño las señales destinadas a tal fin, están nombradas con el sufijo *"_rdy"*, no obstante han sido omitidas de los diagramas para facilitar su legibilidad.

Como se detallará más adelante, en total se dispone de 1536 ciclos de reloj para procesar cada muestra, desde que entra hasta que sale del sistema. No obstante, no es necesario tanto tiempo para realizar todas las operaciones, incluso en el peor caso. La mayor parte de los componentes emplea un número fijo de ciclos para procesar:

- 48 para *audioIO* (24 en la entrada y 24 en la salida).
- 6 para *trackInputMixer*.
- 9 para *trackController*.
- 2 (siempre) más 2 por cada efecto aplicado en la gestión de *fxController*.
- 3 si *fxUnitDelay* está activo.
- 4 si *fxUnitOverdrive* está activo.
- 3 si *fxUnitJamon* está activo.
- 4 si *fxUnitFuzz* está activo.
- 3 si *fxUnitCompresor* está activo.
- 3 si *fxUnitFlanger* está activo.
- 514 si *fxUnitLPF* está activo.
- 514 si *fxUnitHPF* está activo.
- 6 para *outputMixer*.

Las señales de control transmiten pulsos, su función es informar de eventos que tienen lugar en el sistema. En su mayoría las genera el módulo *userControl* (detallado más adelante en la sección 4.8.1), aunque también aparecen en el contexto de la sincronización; es el caso de *tempo* o *tempo_sync*.

Por último las señales de estado sirven para manifestar la situación de los componentes, se asemejan a etiquetas o *flags*. En muchos casos (especialmente en los que contienen información que debe transmitirse al usuario) requieren de una señal adicional para indicar cambios de estado mediante pulsos, de ello depende el funcionamiento del módulo *userDisplay* (más concretamente su componente *btTransmitter*).

Para explicar mediante un ejemplo el funcionamiento de estos dos últimos tipos, supóngase el volumen de una pista. Este debe ser ajustado mediante señales de control, pulsos que indican cuándo incrementa, decrementa o debe cargar una cuantía determinada. El valor de este parámetro, es decir, la salida del registro que lo almacena, constituye una señal de estado. La arquitectura del módulo *Bluetooth* (detallada en la sección 4.9.2) obliga a indicar cuándo un estado ha cambiado, por ello al hacerse efectiva la alteración en el valor del volumen, se genera un pulso alertando de ello. Sin embargo no siempre es necesario, puesto que muchas señales de estado solo funcionan a nivel interno y no se retransmiten al usuario.

4.3. Entrada y salida de audio (audioIO)

Este módulo al ser el primero del flujo de audio, es el que realiza la comunicación con el códec I2S. Los datos se intercambian con él en serie, utilizando una línea por cada dos canales (ya que el protocolo es estéreo).

Como se observa en la figura 4.1, *audioIO* se estructura en tres componentes: *inSampleShifter*, *outSampleShifter* y *mclkGenCounter*.

4.3.1. Relojes I2S

De acuerdo al protocolo I2S, son necesarios tres relojes para sincronizar la entrada/salida en serie: MCLK, SCLK y LRCLK, ordenados de mayor a menor frecuencia. Entre ellos debe haber una relación entera, en este caso se ha utilizado (siendo *fm* la frecuencia de muestreo):

$$MCLK = 256 * fm$$

$$SCLK = 64 * fm$$

$$LRCK = fm$$

Dicha relación se ha establecido considerando que la frecuencia base del reloj es 75 MHz, y a partir de la misma resulta simple generar otro derivado de 25 MHz empleando un contador. Una vez cada tres ciclos este incrementa su valor, por tanto, al tomar los bits

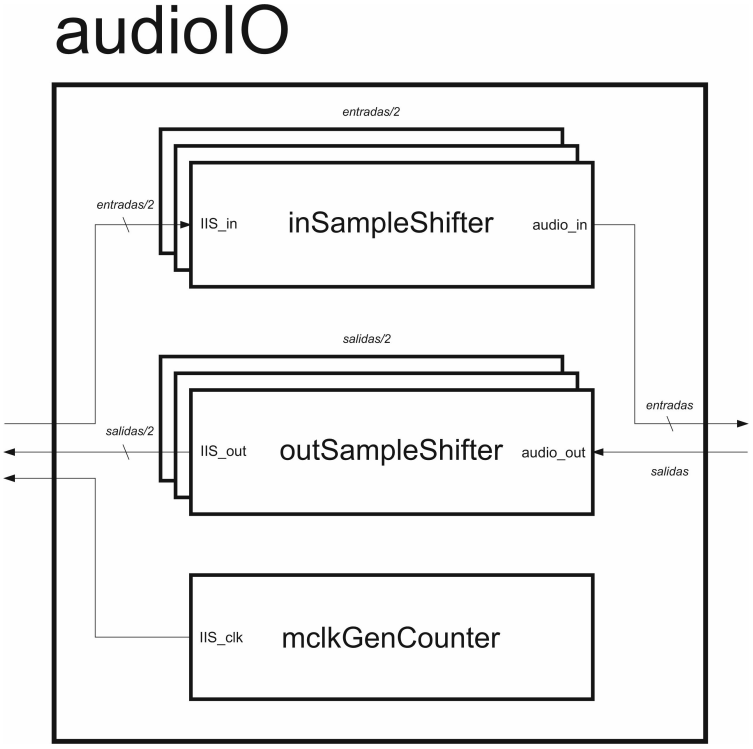


Figura 4.1: Entrada y salida de audio.

pertinentes del mismo es inmediato obtener MCLK (bit 0, 12,5 MHz), SCLK (bit 2, 3,125 MHz) y LRCK (bit 8, 48,828125 MHz). [16–18]

En este diseño, dichos relojes se generan internamente en *mclkGenCounter*, son derivados del reloj principal del sistema.

Una alternativa habría sido delegar la tarea a un reloj externo, generado por el códec I2S. No obstante, esto despertaría la necesidad de sincronizar dos dominios de reloj, suponiendo más complejidad en el diseño y una mayor utilización de recursos *hardware*. Por este motivo se ha estimado más conveniente generar el reloj dentro del sistema.

Tanto la entrada como la salida utilizan el mismo reloj generado. Esto significa que el sonido entra y sale justo al mismo tiempo, cuando dicho reloj lo marca. Asimismo las muestras de salida se corresponden con las últimas muestras que entraron en el sistema, es decir, no hay varias muestras "en vuelo". Por ello el retardo inducido es ínfimo (en concreto 20 microsegundos, considerando que la frecuencia de muestreo es 48828,125 Hz), a pesar de

todas las operaciones que pueden llegar a realizarse sobre el audio.

De cara al diseño, esta restricción implica que para todo el procesamiento se dispone de un número limitado de ciclos de reloj, tantos como haya entre la llegada de una muestra y la siguiente. Considerando la frecuencia de muestreo (48828,125 Hz) y la frecuencia del reloj principal (75 MHz), resulta un total de 1536 ciclos para el procesado.

Aunque sería posible construir una arquitectura que destine más ciclos al procesamiento, afectaría negativamente al tiempo de retardo.

Evidentemente, la velocidad del reloj principal del sistema es superior a la de los relojes I2S.

4.3.2. Paralelización de las muestras de entrada

Puesto que en el estándar I2S los datos se transmiten en serie, es necesario recibir bit a bit cada muestra, mientras se va cargando en un registro de desplazamiento (que actúa como *buffer*). Una vez completada la transmisión, la información se mantiene en dicho registro, hasta que llega la siguiente muestra.

Por cada dos canales de entrada, se precisa un *inSampleShifter*. Esto es debido a que el protocolo I2S es estéreo, pero las entradas del sistema son mono. Gracias a esta optimización se logra duplicar el número de entradas, ya que los instrumentos son típicamente mono: una guitarra, un piano, una flauta, la voz de un cantante... Todos ellos tienen una única fuente de sonido y carece de sentido tenerla repetida dos veces.

En caso de ser necesario sonido estéreo, bastaría con utilizar dos entradas.

4.3.3. Serialización de las muestras de salida

De forma complementaria al módulo anterior, *outSampleShifter* serializa las muestras para enviarlas, acorde al I2S.

Para aprovechar mayor número de salidas, se aplica el mismo concepto de audio mono. Ni los amplificadores de guitarra o bajo, ni los monitores de directo, ni los sistemas PA son estéreo. De este modo el usuario dispone del doble de salidas para configurar según desee.

La implementación de este módulo consta de un registro de desplazamiento, cuyo primer bit conforma la salida en serie. Este se carga con la muestra a transmitir, y cuando corresponde, se desplaza sincronizado con el reloj SCLK.

4.4. Mezcladores de audio

En términos de audio digital, el proceso de mezcla consiste en combinar múltiples muestras de sonido en una sola. La forma de implementarlo es mediante operaciones de suma. En determinados casos también es necesario multiplicar por factores entre cero y uno, para ajustar el volumen del resultado.

4.4.1. El problema del desbordamiento

En la aritmética en punto fijo se utiliza un número constante de bits para representar los valores, por ello, el principal problema que plantea la adición de muestras es el desbordamiento: puede darse el caso en que no sea posible representar el resultado con tal anchura. Es habitual observar como en mezcladores analógicos o digitales el valor de las muestras se divide entre dos para impedir este fenómeno, sin embargo el producto que resulta no es real, pues la amplitud de onda se ve reducida.

La solución por la que se ha optado es utilizar muestras de 32 bits. En un sistema cuyos DACs y ADCs solo son capaces de trabajar con muestras de 24 bits, puede parecer que no tiene sentido trabajar con 32. No obstante, disponer de ocho bits de guarda evita el desbordamiento en gran medida, haciendo posible preservar mayor cantidad de información tras operar con las muestras. De hecho, sería posible sumar 256 veces el máximo valor representable en la entrada (situación que nunca va a darse en la práctica) y la salida del sistema seguiría sin desbordarse.

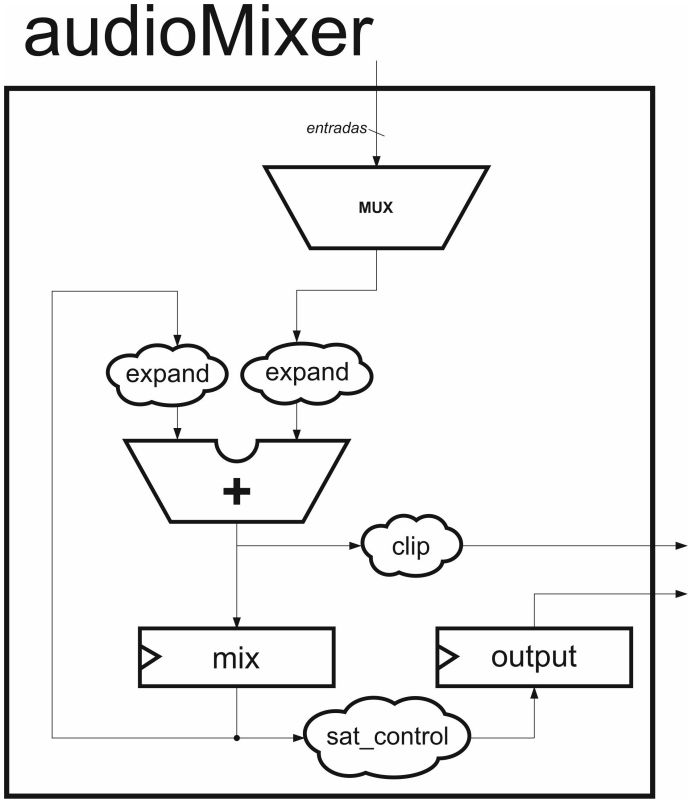


Figura 4.2: Ruta de datos de *audioMixer*.

4.4.2. Módulos de mezcla

Se han diseñado varios módulos de mezcla, en función de lo necesario en diversos puntos del sistema.

AudioMixer

AudioMixer es un mezclador multiciclo que admite múltiples entradas (la especificación en VHDL es paramétrica), seleccionables de cara al proceso de mezcla, es decir, que pueden ser habilitadas o deshabilitadas en tiempo real.

Este módulo ha sido diseñado (principalmente) para su uso en los módulos que comunican con la entrada/salida: *trackInputMixer* y *outputMixer*.

Una vista general de su ruta de datos se proporciona en la figura 4.2.

Mezcla

Cuando se activa la señal *input_rdy* comienza a mezclar, empleando para ello tantos ciclos como entradas de audio tenga el mezclador. El funcionamiento (coordinado por una máquina de estados) consiste en acumular secuencialmente el valor de las muestras, empleando un sumador y un registro. Mediante un multiplexor se seleccionan las muestras a intervenir en la operación de suma acumulada, de una en una; para aquellas muestras excluidas de la mezcla, se sustituye su valor por cero. De este modo se reutiliza el *hardware* empleado para la suma, es decir, se ahorran recursos. Podría haberse optimizado más y emplear solo un ciclo por cada entrada activa (en lugar de reemplazar las entradas inactivas por cero y sumarlas también), sin embargo, no merece la pena destinar CLBs a tal medida, pues el sistema debe funcionar también en el caso más desfavorable (todas las entradas activas).

Para esta situación y todas las relativas a los tiempos de las operaciones realizadas en el audio, ha de tenerse en cuenta lo expuesto en la subsección 4.3.1: el tiempo total de procesamiento no puede exceder el periodo del reloj I2S.

Control de desbordamiento

En caso de detectarse un desbordamiento en el resultado final, se "limita" el resultado (*clipping*), devolviendo el máximo o mínimo valor representable, en función del signo. Además se activará una señal de salida para indicar que el resultado de la mezcla está saturado. Como son varias muestras las que intervienen en el proceso de mezcla, podría darse la situación en que el valor resulte saturado antes de sumar todos los operandos. En tal caso, ya no sería posible conocer el resultado real de dicha mezcla, porque el resto de sumas se realizan sobre un valor "ficticio" (el máximo o mínimo representable). Para prevenir el fenómeno en cuestión, durante el cálculo del total acumulado se aumenta la anchura de las muestras a sumar, añadiendo a las mismas tres bits de guarda. De este modo, para un máximo de ocho entradas es matemáticamente imposible obtener un resultado que sufra saturación. Debe tenerse en cuenta que se utilizan tanto valores positivos como negativos para codificar

las muestras, y la saturación intermedia es susceptible de desaparecer al añadir un nuevo valor a la mezcla. Por ello la limitación solo se aplica al valor final (el total acumulado), lo que permite reducir considerablemente el número de muestras saturadas en la mezcla final, afectando positivamente a la nitidez del sonido y preservación del rango dinámico.

AudioMixerCombi

Como el propio nombre permite deducir, este mezclador es una versión combinacional del anterior, no obstante, presenta una serie de limitaciones.

Este mezclador solo admite dos entradas. Así se evitan violaciones de temporización (puesto que es combinacional) y se previene el uso excesivo de recursos de la FPGA (frente al único sumador del módulo anterior, supondría un "árbol" de operadores).

También dispone de control de desbordamiento, así como de notificación de la misma. Se basa en limitación y devuelve el máximo o mínimo valor representable según proceda, al igual que el módulo anterior (*AudioMixer*).

AudioMixerCombi tiene cabida en operaciones simples de mezcla, como el *overdub* en la grabación de *loops*, o en las pistas, al fusionar el sonido grabado con el sonido entrante.

AudioMixerManualBalance

En el caso de *audioMixerManualBalance*, se cubre la necesidad de obtener una mezcla gradual de dos fuentes. Por ejemplo, el 25 % de una y el 75 % de otra. Este balance se ajusta en tiempo real.

Para ello, se realizan multiplicaciones en punto fijo (Q2.4) de las muestras originales por un factor de reducción. Después se suman al igual que en los otros mezcladores, con capacidad para controlar y detectar el desbordamiento (a pesar de que es prácticamente imposible que tenga lugar en este módulo). Para dicha operación se utiliza directamente un *AudioMixerCombi*.

El proceso de mezcla emplea siempre tres ciclos. Está segmentado por motivos de seguridad en la temporización, puesto que las multiplicaciones son operaciones moderadamente

complejas en términos de *hardware*.

Su funcionamiento lo coordina una FSM, que al recibir las entradas dedica un estado a multiplicarlas por los factores de reducción. Al ciclo siguiente pasa a otro estado destinado a la mezcla. Finalmente registra el resultado y activa una etiqueta para indicar que el proceso ha finalizado.

Este módulo sirve especialmente para controlar ciertos efectos, que requieren mezclar gradualmente una muestra alterada con otra sin modificar.

CompressorMixer

Este mezclador es una variante de *audioMixer* que incluye un amplificador de volumen y un compresor. Su finalidad es servir en el *outputMixer*, tal que las muestras con un volumen demasiado bajo para la salida puedan realzarse, pero sin llegar a saturar, gracias a la acción del compresor.

La figura 4.3 muestra una vista simplificada de su ruta de datos.

De cara a la mezcla, el funcionamiento es idéntico al de *audioMixer*, con diseño multiciclo. No obstante, también realiza la operación de amplificado a las muestras mezcladas, cuyo resultado es posteriormente analizado por el algoritmo de compresión. Así se logra determinar el grado de actuación del compresor.

El compresor comprueba en cada muestra resultante si su valor supera un umbral (establecido por un parámetro de entrada ajustable en tiempo real). Si por cada 256 muestras, más de 50 superan el umbral, aumentará una unidad el factor de compresión (se suma uno a su valor). Si ninguna muestra de las 256 supera el umbral, el factor de compresión se reduce también una unidad (se resta uno a su valor).

Cada ventana de 256 muestras representa 5,24 milisegundos de tiempo (puesto que la frecuencia de muestreo son 48828,125 Hz). Esto garantiza que los incrementos o decrementos en el factor de compresión se realizarán como máximo cada dicho periodo. Por tanto, en el caso más brusco el factor de compresión tardará 83,38 milisegundos en alcanzar su valor máximo.

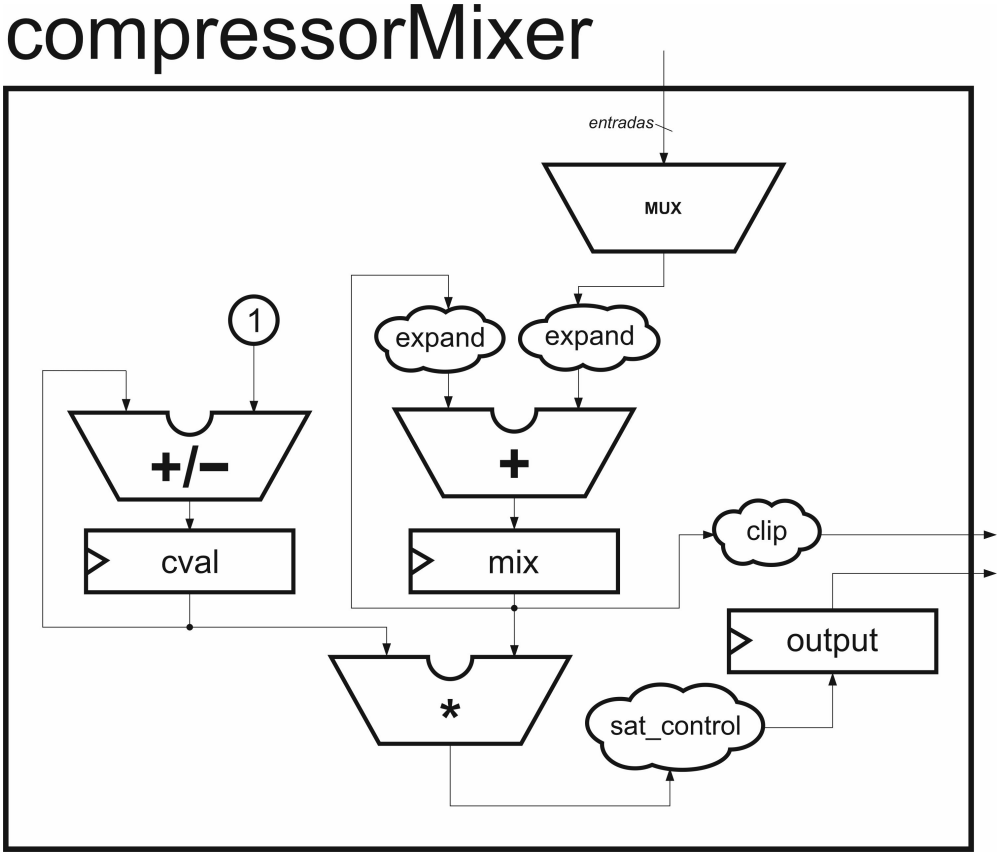


Figura 4.3: Ruta de datos de *compressorMixer*.

Por otro lado 50 muestras de 256 son un 20 % del total. Considerando que las ondas de sonido oscilan y no son regulares, es apropiado establecer este tipo de relación. El efecto de esto es que se ralentizan los incrementos en el factor de compresión, al ser necesario que dicho porcentaje de muestras exceda el umbral.

Teniendo en cuenta los dos párrafos anteriores, el objetivo es obtener una respuesta suave con un ataque moderado, aunque algo más pronunciado cuando el nivel de la señal es elevado.

El amplificado consiste en multiplicar la mezcla por un valor (ganancia), calculado a partir del volumen indicado por el usuario a la entrada del módulo y el factor de compresión determinado por el sistema. De esta manera se unifican las operaciones, tal que solo sea necesaria una multiplicación.

$$ganancia = volumenUsuario - factorCompresion + 1$$

Finalmente, se aplica un limitador combinacional, al igual que en *audioMixer*. Es necesario, a pesar de que el compresor pueda prevenir el desbordamiento, ya que ante picos bruscos en la señal, la acción del compresor no tendría ningún efecto (ni debe tenerlo).

Respecto al *audioMixer*, *compressorMixer* requiere 2 ciclos más para el procesamiento; dado que la máquina de estados principal tiene dos estados más, de un ciclo cada uno, para el amplificado y la compresión. Ambos suceden al estado de mezcla descrito anteriormente para el *audioMixer*, tal que antes de enviarse el resultado a la salida se realiza la multiplicación por la ganancia calculada, y el compresor analiza la muestra final.

4.4.3. Mezcladores de entrada (*trackInputMixer*)

Para utilizar las muestras de entrada en el sistema, es preciso en primer lugar convertirlas a la anchura apropiada. Como se observa en la figura 4.4, antes de ser mezcladas, las muestras de entrada son reajustadas de 24 a 32 bits. Para ello se realiza una operación combinacional que tiene en cuenta el signo de la muestra, que añade ceros si es positiva o unos si es negativa.

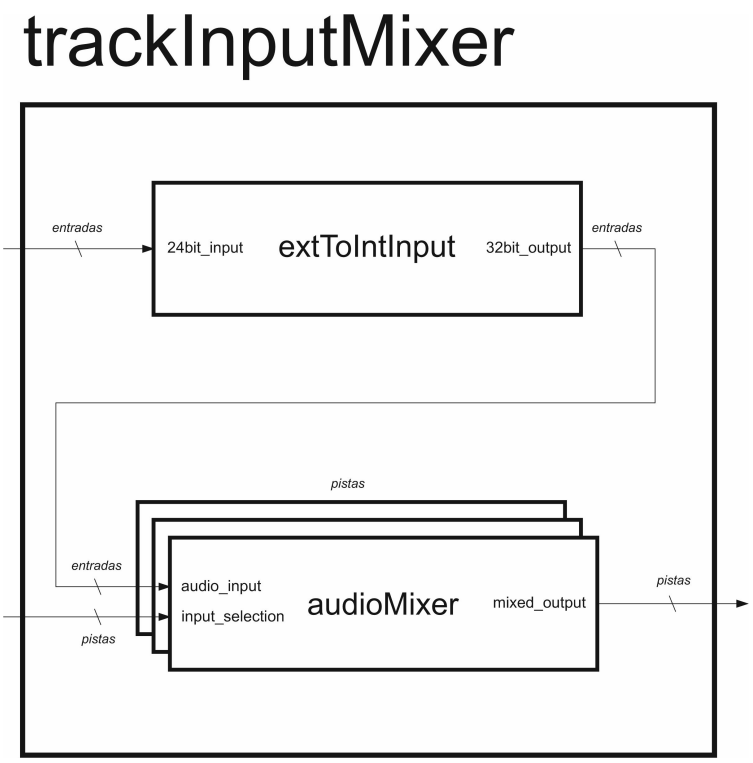


Figura 4.4: Mezclador de entrada

Una vez adaptadas las muestras de entrada, se procede a su mezcla, para enviar a cada pista las correspondientes. Por ello existe un mezclador (*audioMixer*) para cada pista, a cada uno de los cuales se le indica qué entradas deben formar parte de la mezcla.

Puesto que es matemáticamente imposible que la mezcla sature en este punto del sistema, la detección de desboardamiento de los *audioMixer* no se utiliza.

4.4.4. Mezcladores de salida (*outputMixer*)

De forma análoga a *trackInputMixer*, en este módulo se mezclan las muestras procedentes de las pistas (concretamente, de su procesador de efectos asociado), para generar el sonido que ha de enviarse a cada una de las salidas del sistema.

En primer lugar, como se observa en la figura 4.5, un componente *compressorMixer* recibe desde el módulo de control los niveles de compresión y volumen a aplicar en cada canal de salida. Por otro lado, desde las pistas, recibe a qué salidas debe enviar su audio.

Por ejemplo, la pista 1 indica al *outputMixer* que debe sonar por los canales 1 y 2, la pista 2 solo por el canal 2... etcétera.

Las mezclas resultantes (una por cada salida) deben reducirse en anchura de 32 a 24 bits, tal que sean aptas para el códec de audio del sistema. Este proceso se realiza de forma combinacional en *intToExtOutput*. Para garantizar la máxima fidelidad, las muestras se someten a redondeo y saturación. El redondeo se implementa truncando los últimos ocho bits de la señal original, si la muestra es positiva y el más significativo de ellos es uno, se suma a los 24 restantes; si la muestra es negativa y el mismo bit es cero, se resta uno. En caso de que dicha suma desborde, se aplica limitación reemplazando su valor por el máximo o mínimo representable (según corresponda).

Por otro lado, con vistas a proporcionar al usuario información sobre el estado del proceso, el componente *outputLevel* mide la amplitud de las muestras. En conjunto con el medidor, que es combinacional, la implementación consta de una máquina de estados. Esta se encarga de registrar periódicamente el valor más alto manifiestado por *outputLevel* en el intervalo de tiempo correspondiente a 4096 muestras de audio. Cuando no hay ninguna salida activa, la máquina de estados queda "en espera". Este mecanismo es imprescindible porque los cambios registrados en el volumen de salida se notifican a los módulos que retransmiten al usuario el estado del sistema; enviar el nivel de salida de todas las muestras procesadas no solo es innecesario, sino que también sobrecargaría la comunicación (especialmente en lo que respecta al módulo *Bluetooth*). La longitud de los intervalos (4096 muestras) equivale a 83,9 milisegundos porque permite alcanzar un equilibrio entre fluidez (pues con periodos más largos aparecen "saltos" al visualizar el estado del nivel de salida) y número de mensajes enviados al controlador remoto.

Además el *compressorMixer* también indica al exterior del módulo si está comprimiendo la salida, muy útil e incluso necesario para tener control sobre el sonido resultante. De este modo, se puede saber cuando el volumen es demasiado alto.

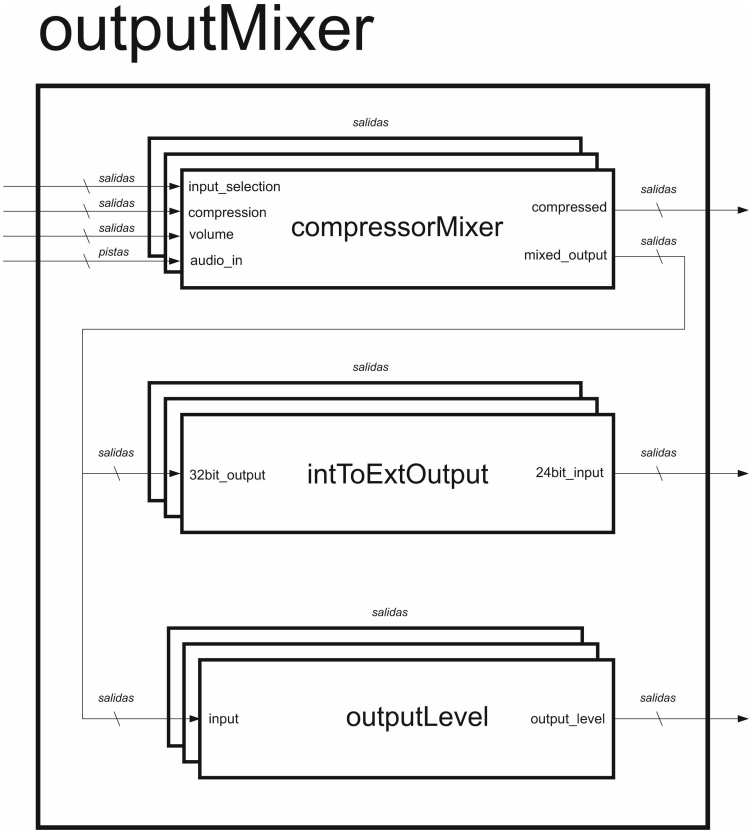


Figura 4.5: Mezclador de salida

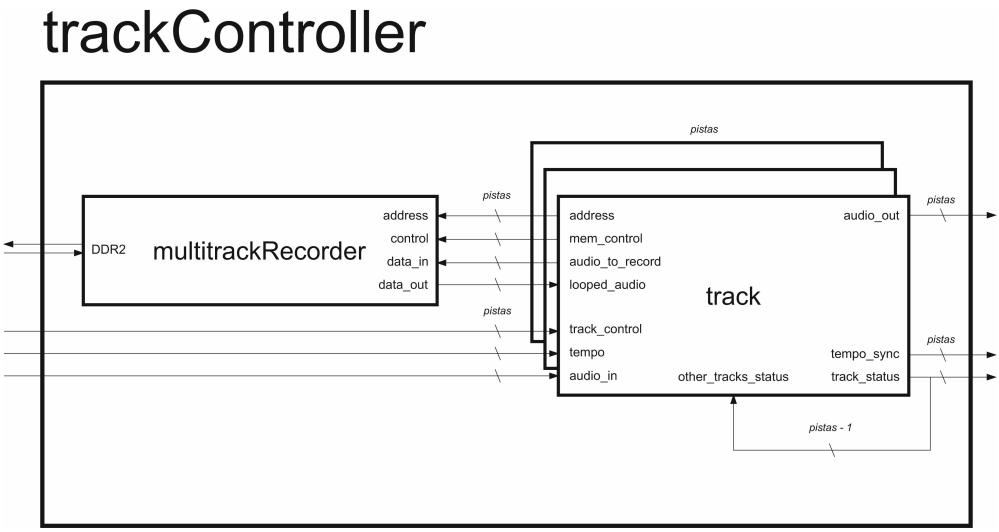


Figura 4.6: Controlador de las pistas

4.5. Pistas

En las pistas se realiza el control de la reproducción y grabación, y por tanto del acceso a memoria. También gestionan la sincronización de los *loops*. Por ello las operaciones lógicas más complejas tienen lugar en estos módulos.

4.5.1. Controlador de las pistas (*trackController*)

El componente que alberga las pistas y el controlador de memoria es *trackController*.

Las pistas reciben el audio de entrada, ya mezclado en *trackInputMixer*, así como el audio previamente grabado en memoria. Para obtener este último, las pistas mantienen comunicación con *multiTrackRecorder* (el controlador de memoria).

Cada pista recibe además sus correspondientes señales de control y el *tempo*, para la sincronización.

Por otro lado, las pistas retransmiten su estado entre ellas (también para sincronización) y al resto del sistema.

4.5.2. Pista (*track*)

Las ocho pistas son los ocho corazones del *LoopMAN*. Su arquitectura se compone de múltiples módulos y máquinas de estados sincronizadas, la figura 4.7 muestra un esquema.

Teniendo en consideración la labor principal de las pistas, estas gestionan la grabación y reproducción de audio, así como su sincronización. Por motivos de complejidad en el diseño e integración de dichas funciones, se han delegado las tareas en diversos componentes.

TrackFlow

Acorde a otras secciones de esta memoria, la descripción del sistema atiende al orden en que el audio lo atraviesa, no obstante, en las pistas es más complejo definir este flujo. En cualquier caso, el módulo principal es *trackFlow* y consta fundamentalmente de una máquina de estados, la cual coordina la acción de los otros componentes. Cuando llega una muestra nueva, comienza a funcionar: sale del estado de espera y pasa a intercambiar los datos pertinentes con la memoria, es decir realiza la lectura y si corresponde, la escritura. Gracias al diseño altamente optimizado del controlador de memoria, la operación anterior se resuelve en un solo ciclo. A continuación establece las entradas del mezclador interno, para fusionar el sonido entrante con el leído de memoria. Por último, el resultado anterior se somete (combinacionalmente) a las operaciones de ajuste de volumen y la muestra final es almacenada en el registro de salida.

El mezclador (*audioMixer*), es el mismo que el expuesto en la sección 4.4.2. Su salida se somete a un atenuador de audio (*audioDimmer*), módulo que realiza una multiplicación en punto fijo (Q2.4) para reducir el volumen final. Se ha seguido una idea habitual en mezcla de audio digital, tal que solo sea posible atenuar y nunca amplificar el volumen de las pistas; de este modo se obliga a trabajar con muestras de amplitud adecuada y se evita el desbordamiento, así como la pérdida de fidelidad.

En la última étapa, la salida de audio se silencia, si el *mute* ha sido activado, o bien si la señal *not_ alone* se encuentra a alta, pues eso supone que alguna otra pista está en modo

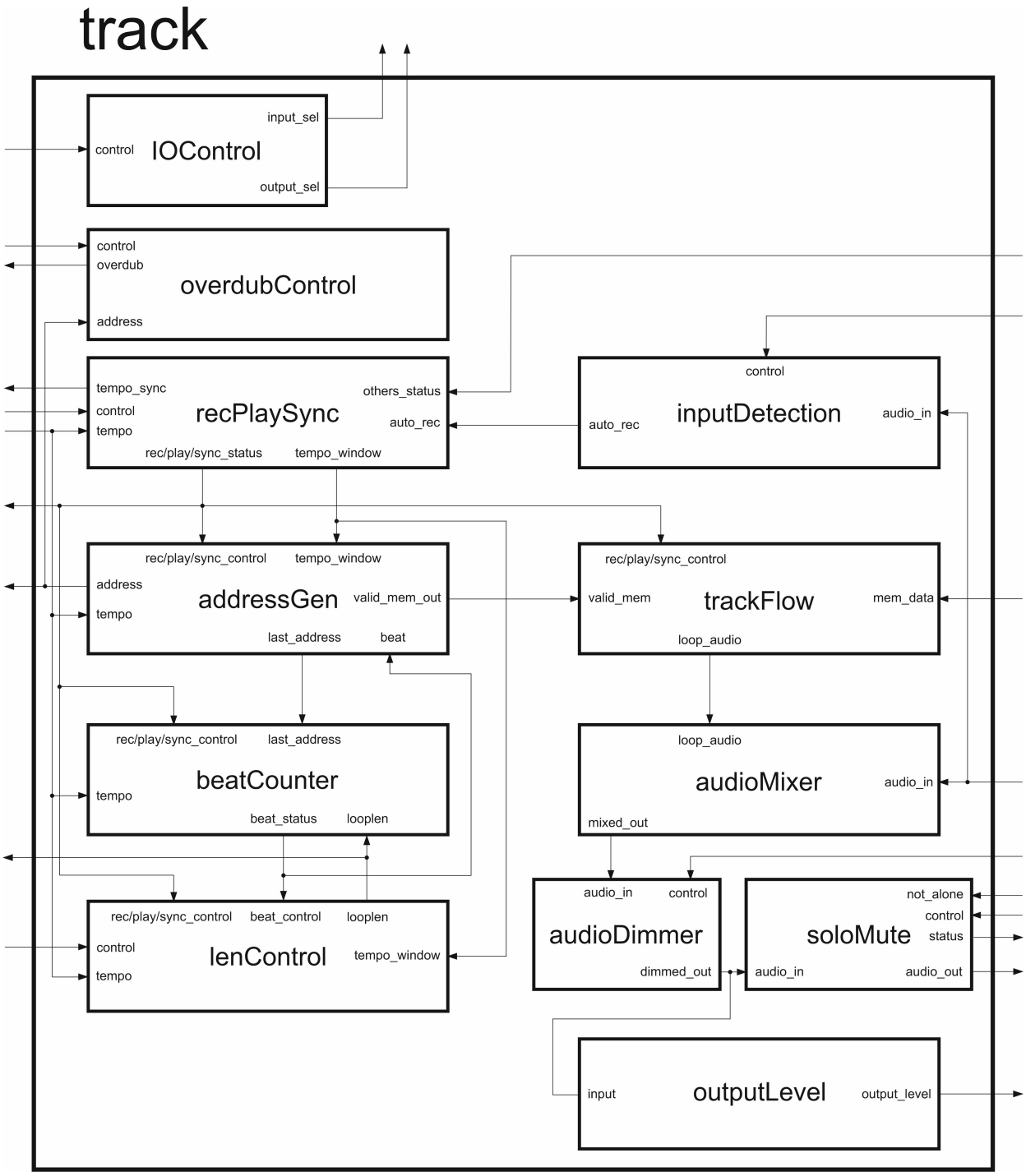


Figura 4.7: Pista

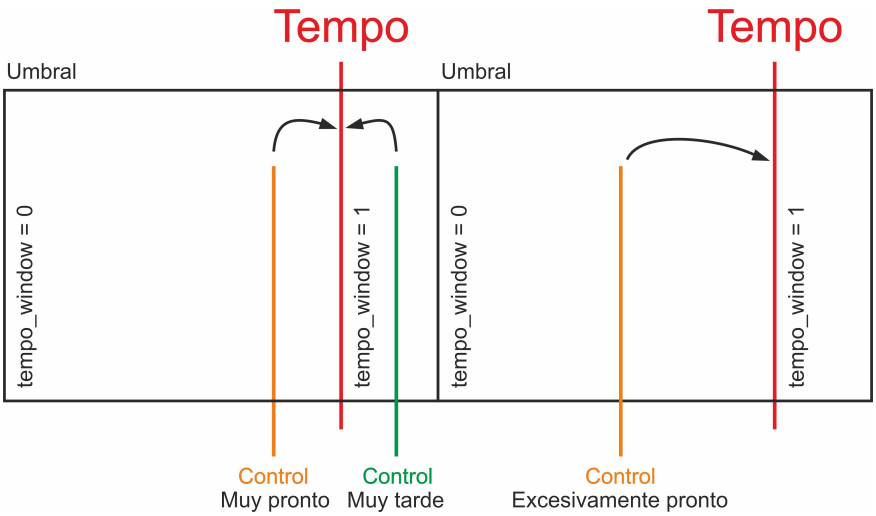


Figura 4.8: Se observa el efecto de la sincronización en las señales de control.

solo.

RecPlaySync

TrackFlow actúa acorde a condiciones marcadas por otros módulos, como es el caso de *recPlaySync*, encargado de sincronizar las señales de control (*rec* y *play*) respecto al *tempo*. La sincronización está condicionada a que la opción *sync* esté activada y al estado de las otras pistas, recibido por la señal *others_working*.

Al presionar el botón de grabar o reproducir, es inevitable que los humanos tardemos o nos adelantemos respecto al *tempo*: siempre se producirá un desfase. Por ello, se ha establecido una medida para las órdenes adelantadas, que consiste en no activar la señal correspondiente (grabar o reproducir) hasta que llega el siguiente pulso de *tempo*. En el escenario contrario, cuando la pulsación se realiza ligeramente después respecto a al *tempo*, se mide el desfase; si este no excede un umbral (0,167 segundos, que equivalen a 1,25 millones de ciclos), la señal de control se activa de inmediato. Este comportamiento está ilustrado en la figura 4.8. Para que otros módulos del sistema puedan respetar la sincronización en este segundo caso, *recPlaySync* escribe en la señal *tempo_window*.

Autorec

Tal y como se ha descrito en la sección 2.1.3, el sistema dispone de una función *"autorec"*. El componente *inputDetection* realiza la misma, para ello monitoriza el nivel de entrada (combinacionalmente) y genera un pulso cuando se detecta que al menos quince muestras exceden en amplitud un valor prefijado (que equivale al 6,7 % del valor máximo). El motivo de ello es evitar que el *"autorec"* se active accidentalmente por el ruido u otros factores similares.

AddressGen

En paralelo, *addressGen* genera las direcciones para el acceso a memoria. También mantiene el control del rango de direcciones escritas, es decir, aquellas que contienen muestras que pueden reproducirse.

Además, este módulo de acceso a memoria también implementa medidas de sincronización. En la situación de desincroncización antes mencionada, cuando la orden llega al sistema con retardo respecto al *tempo*, cabría esperar que la reproducción no comenzase desde el principio, sino desde el instante que correspondería en caso de que la orden estuviese perfectamente sincronizada con el *tempo*. Pues bien, este ajuste se realiza automáticamente. Es por esto que *addressGen* recibe la señal *tempo_window*.

La implementación de *addressGen* consta de una FSM. Su primer estado es de espera, solo avanza a uno de los siguientes cuando llega una nueva muestra. En tal caso existen dos posibilidades, por un lado el estado de sincronización (si no está reproduciendo ni grabando), que avanza el puntero de acceso para sincronizar las pistas ante una posible pulsación tardía de la opción *play* o *rec* (como se ha explicado en el párrafo anterior). Permanece en este último hasta que expira *tempo_window* o bien comienza la reproducción o grabación. Por otro lado, si al llegar la nueva muestra ya estaba reproduciendo o grabando, pasa a un estado que comprueba si la dirección actual está en el rango válido. Así se previenen las lecturas de datos que no correspondan. Al siguiente ciclo pasa a otro estado que actualiza la dirección

para la próxima muestra, es importante tener en cuenta que este proceso está sincronizado con la máquina de estados principal *trackFlow*), tal que el acceso a memoria se completa antes de cambiar la dirección. Por último, cuando el usuario envía una orden de borrado a la pista, se reinicia el rango de direcciones válidas, de esta manera queda implementada la eliminación del audio grabado.

OverdubControl

Otro componente estrechamente relacionado con la memoria es *overdubControl*, que indica a la memoria el grado de atenuación de las muestras antiguas (establecido por el usuario), cuando se graba encima de ellas. La primera vez que se graba en un rango de direcciones el *overdub* debe ser necesariamente cero, en caso contrario el audio se mezclaría con ruido. Por este motivo, el componente necesita conocer las direcciones de acceso a memoria.

LenControl y BeatCounter

Para establecer (en pulsos) la duración del *loop*, el componente *lenControl* mantiene y actualiza dicha información.

BeatCounter está constituido por otra máquina de estados, que lleva cuenta del *beat* (pulso) actual. Si el estado del sistema es el apropiado (está grabando o reproduciendo y la memoria no está vacía), cada vez que llega un pulso de *tempo* se incrementa el contador. También dispone de un estado de *reset* que establece a cero el contador.

La cuenta del pulso es relevante en otros componentes de la pista, como es el caso de *addressGen*, pues al alcanzar el valor máximo de cuenta, es necesario que el puntero de acceso a memoria vuelva al inicio.

OutputLevel

El volumen de salida de la pista es monitorizado del mismo modo que en *outputMixer*, tal y como se detalla en la subseción 4.4.4.

IOControl

Al margen de todo lo demás, el componente *IOControl* genera las señales que indican a *trackInputMixer* y *outputMixer* la selección de canales de entrada y salida para la pista.

4.5.3. Acceso a memoria (*multiTrackRecorder*)

La solución al problema del acceso a la memoria RAM externa ha sido fundamental para garantizar que el diseño minimiza el número de ciclos consumidos desde que el audio entra hasta que sale. En todo momento hay que tener en cuenta que este es un sistema de tiempo real, y la principal restricción de temporización (en términos globales) viene dada por la frecuencia de muestreo del códec de audio (como se detalla en la subsección 4.3.1).

Las pistas deben acceder a la memoria para almacenar las muestras entrantes, o bien para leer los datos anteriormente escritos. Puesto que en el sistema hay ocho pistas, sería necesario un árbitro que gestionase en qué orden realiza cada pista los accesos a memoria. También debe tenerse en cuenta que es más de una operación por pista, por ejemplo, para el *overdub* primero se lee el dato, después se mezcla y finalmente se vuelve a escribir.

En resumen, por cada muestra que entra al sistema se realizan multitud de transacciones de memoria, tantas que no es viable completar todas y terminar a tiempo el procesamiento de las muestras (antes de que llegue el siguiente dato). Aun suponiendo que fuese posible cumplir con la temporización, la escalabilidad sería nula: no habría forma de pasar a dieciséis pistas, ni de añadir multitud de efectos después, ya que estos también necesitan ciclos para el procesado.

Hay muchos factores que afectan negativamente a la situación. En primer lugar, solo se dispone de una memoria, con un único puerto de entrada y otro de salida (de datos). Esto impide mantener comunicación paralela.

Por otro lado, solo se necesitan datos "pequeños", concretamente 32 bits, por tanto a priori no sirve que la memoria proporcione palabras de 128 bits [1]. Además las ocho lecturas

[1] En la configuración utilizada para la memoria, el controlador proporciona palabras de 128 bits, aunque trabajaría con palabras de 64 si la velocidad del reloj fuese mayor.

se realizan de lugares dispersos de la memoria, pues cada pista lee en una región distinta.

Puede surgir la idea de "alinear" las pistas, tal que cada palabra contenga información de varias pistas, sin embargo no es válido, ya que las pistas no acceden al mismo tiempo a las grabaciones; es decir, que van desfasadas (salvo que empiecen a la vez y tengan el mismo número de compases).

Sin embargo hay dos detalles que favorecen el panorama: cada pista solo realiza (como máximo) una escritura y una lectura por muestra recibida. Y por otra parte, las pistas individualmente barren regiones contiguas de memoria, y siempre en el mismo sentido.

En base a todo lo expuesto, se ha elaborado un diseño para permitir que las ocho pistas (o las que proceda, si se desea extender el sistema) sean capaces de acceder de forma simultánea a la memoria (es decir, todas leyendo y escribiendo en el mismo instante), empleando para ello solamente un ciclo de reloj.

La forma de lograr esto ha sido un mecanismo de *caches* optimizado exclusivamente para este propósito.

Interfaz de acceso a memoria

La arquitectura de *multiTrackRecorder* (ver figura 4.9), presenta en primer lugar, un componente *ram3DDR*. Se trata de un envoltorio para facilitar la comunicación con el controlador de memoria. Hace transparente el envío de comandos para las transacciones, tal que el operar con la memoria DDR2 sea comparable a acceder a una *block ram* más lenta y con una señal de *ACK*. Internamente el módulo utiliza un *IP soft core*: *Memory Interface Generator (7 Series)*, versión 4.1 [9, 19].

Este *IP core*, el *Memory Interface Generator* produce una señal de reloj para la DDR2 (establecido a 300 MHz en este sistema), sin embargo el controlador funciona internamente a una frecuencia que guarda una relación 1:4 o 1:2 con la generada para la RAM. Elegir una u otra condiciona el tamaño de las palabras a utilizar en las operaciones de lectura y escritura.

Lo más conveniente se estimó en emplear la relación 1:4, es decir 75 MHz. La decisión

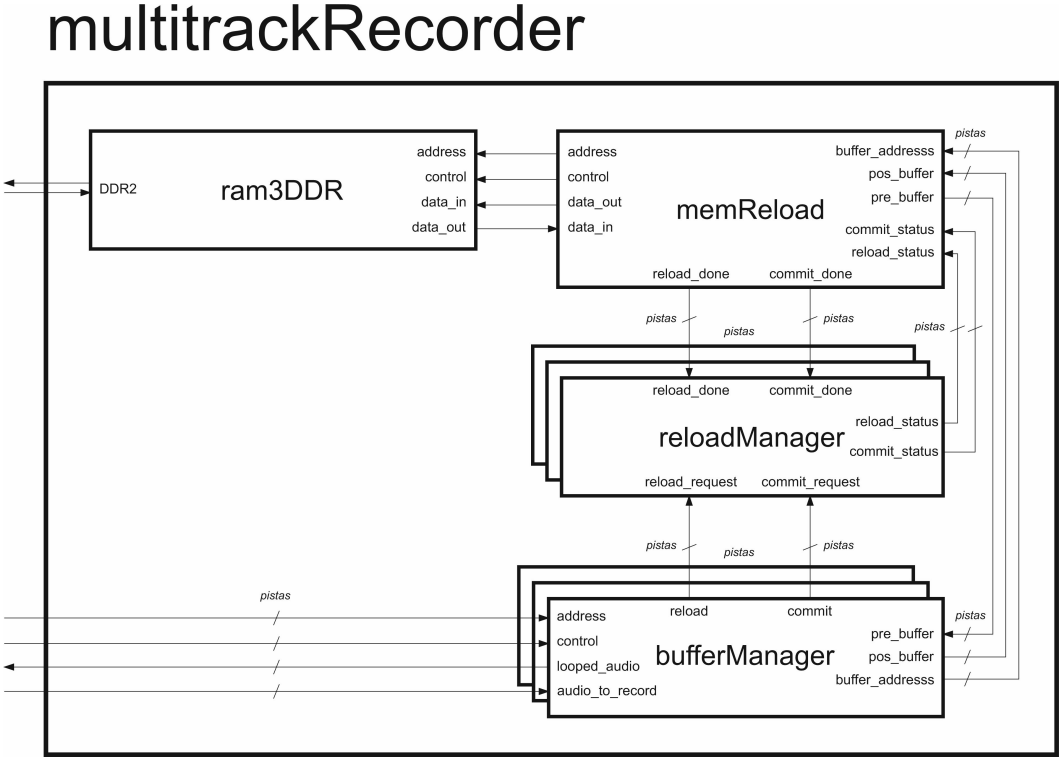


Figura 4.9: Módulo de acceso a memoria

se fundamenta en que el controlador de memoria debe compartir el reloj con el resto del sistema para que no sea necesario sincronizar dos dominios de reloj. Además, de esta manera se admiten operaciones combinacionales más complejas y el consumo del diseño será menor (frente a tiempos de ciclo más cortos). Por otro lado, las palabras que intervienen en los accesos a memoria son más anchas (128 bits frente a 64), lo cual beneficia al carácter contiguo del audio almacenado, siendo menos frecuentes las operaciones de recarga de los *buffers* (como se detalla en la sección inmediatamente posterior). La única desventaja es que se dispone de menor número de ciclos para el procesamiento.

Gestión de los *buffers*

Existen también, por cada pista, dos *buffers* de escritura y dos de lectura: *rd_ buffer*, *wr_ buffer*, *pre_ buffer* y *pos_ buffer*. La anchura de estos *buffers* es igual al tamaño de los datos (las muestras), es decir, 32 bits. Su profundidad es el número de muestras que

caben en una palabra de memoria (cuatro). Para facilitar las operaciones sobre ellos, están implementados como registros de 128 bits.

Asimismo, para realizar lecturas, las pistas indican al *bufferManager* la dirección a la cual desean acceder. El contenido de esa dirección debe estar previamente almacenado en *rd_ buffer*. Cuando se accede a un dato no presente en *rd_ buffer*, se notifica a *reloadManager* para que active el *flag* de recarga asociado al *buffer* en cuestión. Como los accesos son secuenciales es fácil predecir la próxima dirección, por lo que sus datos habrán sido previamente almacenados en el *pre_ buffer* y pasan automáticamente a *rd_ buffer*. El *pre_ buffer* debe ahora cargar desde memoria la dirección siguiente, y *memReload* (que realiza dicha tarea), lo sabe gracias al flag de recarga antes activado. De forma paralela y totalmente transparente a los demás módulos, *memReload* refresca los *pre_ buffer*, ocultando el carácter secuencial del acceso a memoria.

Las operaciones de escritura son realizadas de forma complementaria: *wr_ buffer* almacena los datos escritos. Cuando está lleno, su contenido se copia al *pos_ buffer*, y se activa el *flag* correspondiente. Cuando *memReload* pueda, escribirá en la dirección de memoria correspondiente los datos de *pos_ buffer*.

El único problema que presenta este sistema es la incapacidad para proporcionar al ciclo siguiente los datos de la primera lectura tras saltar a una dirección no contigua. Esto se debe a que el predictor de dirección siguiente falla. Este fallo es asumido por el componente que recibe los datos, es decir el *addressGen* de las pistas (sección 4.5.2). Como el único salto entre direcciones no contiguas se produce al final del *loop* (es decir al volver al inicio de la grabación), resulta inmediato detectar la situación y basta con descartar las siguientes cuatro muestras leídas del *buffer*. Para ello, *addressGen* simplemente excluye dichas primeras muestras del rango de direcciones válidas.

Mapa de memoria

Asimismo, cada pista tiene preasignado un rango fijo de direcciones en memoria. En la figura 4.10 se muestra el mapa de memoria asociado al sistema. Las direcciones tienen 23

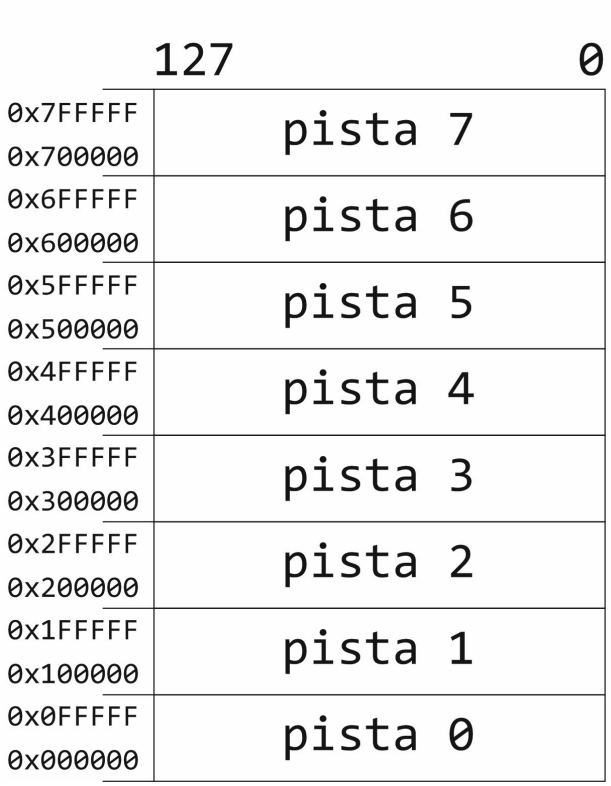


Figura 4.10: Mapa de memoria.

bits, los tres más significativos están asociados a la región correspondiente a cada pista; mientras que el resto se utilizan para seleccionar las palabras de 128 bits.

Cada palabra contiene por tanto, cuatro muestras de 32 bits cada una.

Overdub

Por último, la función *overdub* (que permite mezclar una grabación antigua con una nueva en la misma dirección de memoria) se realiza dentro de *bufferManager*, en un componente llamado *overdubber*. La operación consiste en atenuar las muestras del *buffer* de lectura (*rd_buffer*) y mezclarlas con los datos de entrada, antes de almacenarlos en el *buffer* de escritura (*wr_buffer*).

La atenuación se realiza mediante una multiplicación en punto fijo (Q2.4), utilizando un

componente *audioDimmer*. Para la mezcla se utiliza un *audioMixerCombi*.

4.6. Control de *tempo*

4.6.1. *Tempo*

Recordando el concepto de *tempo*, este hace referencia a la velocidad de interpretación. Se mide en pulsos por minuto (bpm, *beats per minute*).

Es necesario mantener esta noción en el sistema por múltiples motivos. Los detalles se encuentran en la sección 2.1.3, donde se habla de las funcionalidades ofrecidas. De cara a la arquitectura, el objetivo principal es mantener la sincronía internamente entre las pistas, y externamente con el intérprete u otros instrumentos digitales.

En términos de *hardware*, el *tempo* funciona de forma similar a un reloj derivado del principal, generando pulsos a una frecuencia menor, sin embargo no es tal cosa. El *tempo* es en realidad una señal de control, implementada como las señales de carga de los registros: se mantiene a alta durante un ciclo cuando corresponde.

Por ejemplo, el *beatCounter* (módulo que mantiene el estado sobre la reproducción) avanza un pulso (o *beat* en inglés) cada vez que la señal de *tempo* conmuta a alta.

En el siguiente apartado, se describe la arquitectura interna del componente que gestiona el *tempo* del sistema.

4.6.2. *TempoGenerator*

TempoGenerator es el componente que genera la señal principal de *tempo*, en base al control del usuario y mecanismos de sincronización. Como se observa en la figura 4.11, consta de tres módulos.

El módulo homónimo (*tempoGenerator*), mantiene un contador de ciclos que vence al alcanzar el valor indicado por la señal *tempo_ticks*. Entonces generará un pulso. En caso de recibir una petición de sincronía (señal *tempo_sync*), reiniciará el contador y generará también un pulso.

TempoValueSetter establece el valor máximo de la cuenta realizada en *tempoGenerator*. Dicho valor puede ser ajustado directamente mediante señales de control, o bien a través de un mecanismo que detecta la cadencia con que el usuario presiona un botón.

Este último se encuentra en el módulo *tempoMeasurer*, implementado con una máquina de estados. Para detectar la frecuencia de las pulsaciones en el ajuste automático de *tempo*, se mide con un contador el número de ciclos transcurrido entre cada accionamiento del botón. En ello se emplean dos estados: uno de espera y otro de cuenta, tal que el primero transiciona al segundo después de la primera pulsación. Si el contador alcanza su valor máximo, el estado de cuenta retorna al de espera; si por el contrario se acciona nuevamente el botón, se almacena y procesa el valor actual de la cuenta.

En un nuevo estado, se calcula la media total con cada valor obtenido, que determinará el valor final a emplear para generar el *tempo*. De este modo se mitigan los efectos del error humano, puesto que es prácticamente imposible marcar el *tempo* con absoluta precisión. Se ha pretendido dar más "peso" al último valor calculado que a los anteriores, puesto que es frecuente que el usuario sea más preciso después de sucesivos toques al botón. Por ello la fórmula empleada en este cálculo es la que sigue:

$$m(n) = \frac{c(n) + m(n-1)}{2}$$

Donde m es el valor medio calculado y c el valor de cuenta medido entre las dos pulsaciones del usuario.

Este proceso de cálculo de la media se reinicia (es decir, se elimina la última media calculada y comienza de nuevo) en varios escenarios. El primer caso es que se ajuste el *tempo* manualmente mientras se está midiendo la frecuencia de las pulsaciones. Otra situación se da cuando el usuario cambia sustancialmente el ritmo, acelerándolo o ralentizándolo más de un 5 %. Este desfase se calcula multiplicando la media por 0,95 y 1,05 para obtener un límite inferior y otro superior respectivamente:

$$límite\ inferior = m * 0,95$$

$$límite\ superior = m * 1,05$$

En caso de que el valor de cuenta obtenido exceda alguno de dichos límites se considerará un cambio sustancial en el *tempo* y por tanto el cálculo de la media será reiniciado.

La implementación de este mecanismo se ha realizado con multiplicaciones en punto fijo (Q2.4) por las constantes "001110"(0,9375) y "010001"(1,0625).

Establecer el desfase máximo al 5 % se debe a un criterio meramente musical. Por ejemplo, suponiendo que el *tempo* sea 120 bpm ralentizarlo un 5 % resulta en 114 bpm y empieza a ser claramente apreciable.

4.7. Efectos

Los efectos están implementados como un componente separado de las pistas, siguiendo las líneas de la arquitectura modular.

Para optimizar el rendimiento y permitir que el sistema pueda escalar (al introducir más efectos u otros componentes en el flujo principal del sonido) respetando los límites de temporización entre muestra y muestra, este módulo se ha diseñado de forma paralela tal que los efectos se aplican simultáneamente en las ocho pistas.

4.7.1. Controlador de efectos (*fxController*)

Para respetar la jerarquía del diseño, el controlador de efectos es un envoltorio de los ocho procesadores de efectos (uno por pista). Se observa su sencillez en la figura 4.12.

Del mismo modo que otros componentes que forman parte del flujo del sonido (conjunto de módulos que atraviesan las muestras de audio en el sistema), presenta una serie de

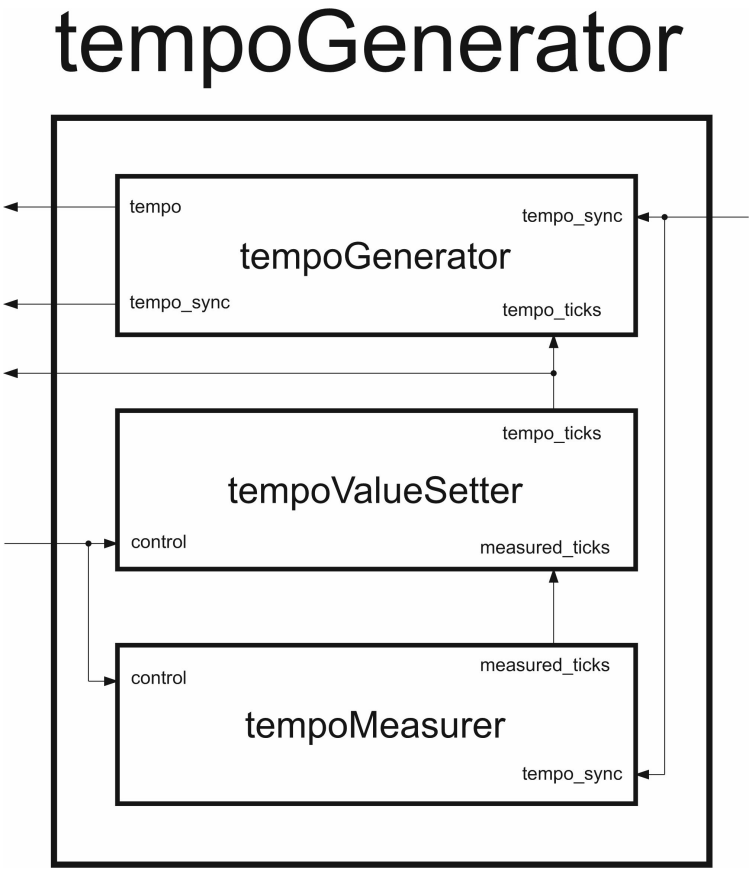


Figura 4.11: Controlador central de *tempo*.

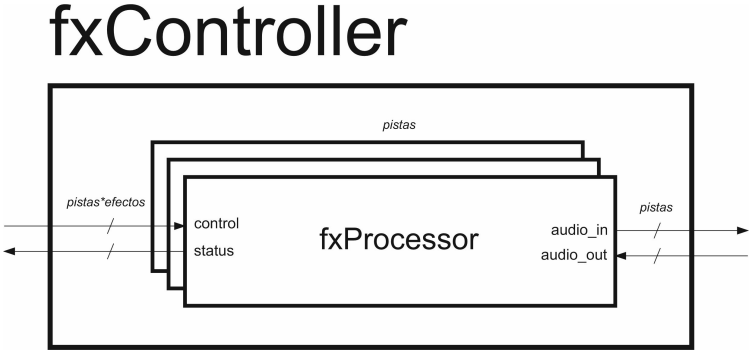


Figura 4.12: Controlador de efectos.

entradas para controlar su comportamiento y salidas para manifestar su estado actual. Por cada pista del sistema, hay tantas señales de control y estado como efectos disponibles.

Evidentemente, el módulo presenta también entradas y salidas de audio (una por pista).

4.7.2. Procesador de efectos (*fxProcessor*)

Para lograr que el sistema pueda aplicar múltiples efectos de sonido a la misma muestra y en cualquier orden, es necesaria una arquitectura como la presente en la figura 4.13.

El control de los efectos (gestionar si están activos o no, así como el valor de sus parámetros) lo realiza *enableControl*.

FxUnit son los módulos que aplican efectos al sonido, por ello reciben una señal desde *enableControl* para regular su comportamiento. Presentan una entrada y una salida para el audio.

Para controlar el orden en que se aplican los efectos, el módulo *queueControl* registra las solicitudes de encolado de los efectos (procedentes de *userControl*). Para ello mantiene una estructura similar a una fifo, que almacena etiquetas asociadas a cada *fxUnit*.

Asimismo, cuando llega una nueva muestra para procesar, empieza a funcionar la máquina de estados finitos presente en *mainControl*. En el registro *fx_ buffer* se almacena el audio recibido. Si hay algún efecto activado y encolado, pasa al siguiente estado, donde se lee en orden la fifo de *queueControl*. Si la etiqueta leída corresponde a un efecto activo, se le envía a este el contenido de *fx_ buffer* y avanza al siguiente estado (de espera). Cuando dicho

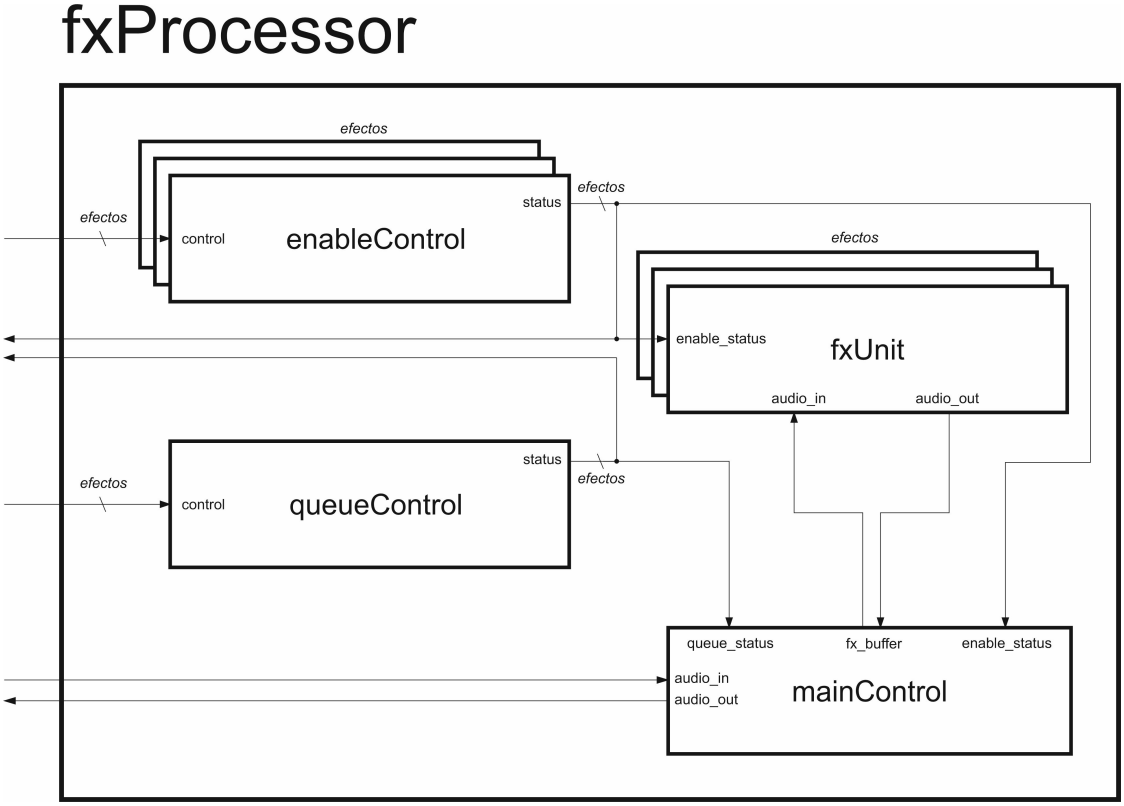


Figura 4.13: Procesador multiefectos.

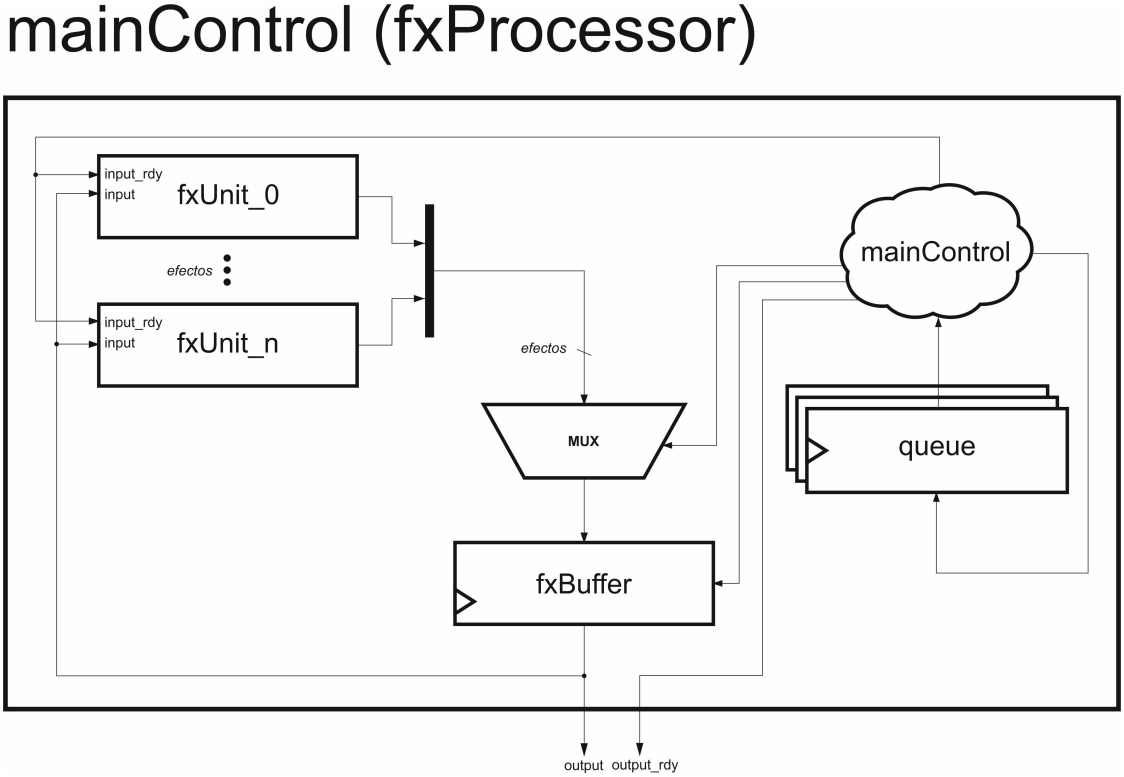


Figura 4.14: Aplanado del módulo mainControl en *fxProcessor*.

efecto finaliza su operación, almacena el resultado de nuevo en *fx_ buffer*, y vuelve al estado anterior para obtener de la fifo el siguiente efecto encolado y activo. Este proceso se repite hasta que no quede nada por leer en la fifo, entonces la muestra final con todos los efectos ya aplicados se transmitiría al exterior (es decir, al siguiente módulo, que es *outputMixer*).

Como apoyo a la descripción anterior, en la figura 4.14 se puede observar el *hardware* correspondiente a la implementación de *mainControl*. El banco de registros *queue* pertenece al módulo *queueControl*. Los *fxUnit* son los distintos efectos de sonido.

4.7.3. Efectos de sonido (*fxUnit*)

En esta sección se describe la arquitectura interna de los efectos de sonido. Para implementar muchos de ellos se han realizado previamente implementaciones en *software* (empleando *Matlab*), para verificar los diseños de un modo más ágil [1, 23–25].

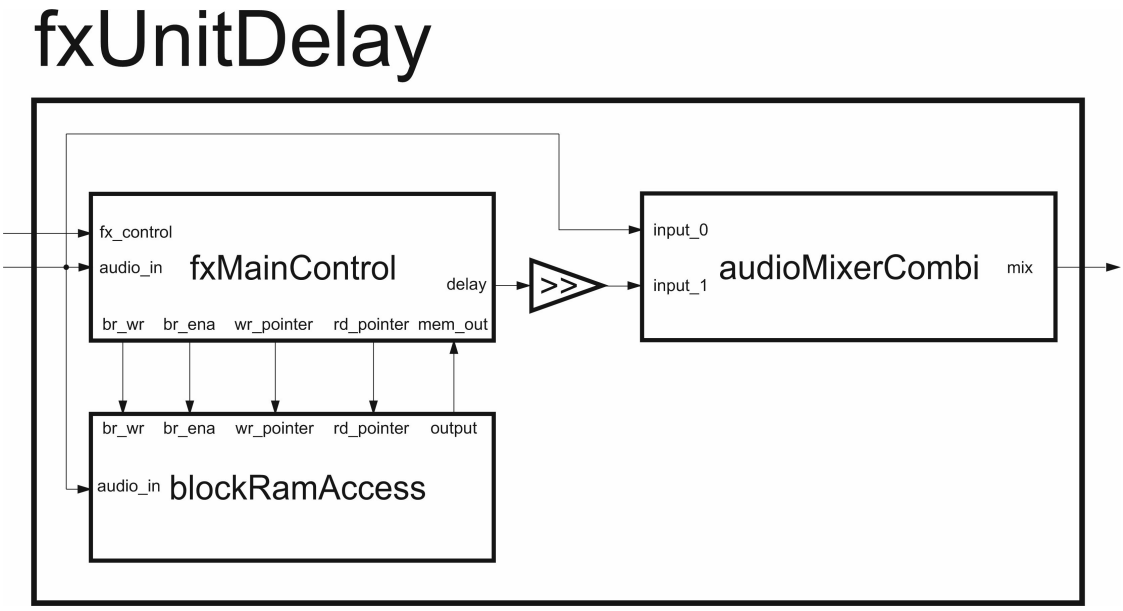


Figura 4.15: Arquitectura de *fxUnitDelay*.

Delay

El *delay* repite el sonido periódicamente en intervalos fijos de tiempo, cada vez con mayor atenuación; por tanto en su implementación digital es imprescindible al menos una memoria, un controlador para gestionar el acceso a la misma y operadores para disminuir el volumen y mezclar el sonido retardado con el actual.

La figura 4.15 muestra un esquema arquitectónico de este componente.

El controlador que coordina la operación del efecto es *fxMainControl*, que consta de una máquina de estados. En primer lugar, espera a que se complete la lectura de la memoria. Después, el volumen de la muestra leída se reduce combinacionalmente a la mitad. Lo obtenido se mezcla con la entrada actual y tras un ciclo de reloj cambia al siguiente estado. Finalmente, mientras el resultado se transmite al exterior también se almacena en memoria, todo a la vez; de esta manera se consigue ahorrar un ciclo. En el último estado también se actualiza el puntero de escritura.

Para implementar el almacenamiento se ha empleado una *block ram*, puesto que la DDR2 disponible en la placa se ha destinado en su totalidad a las pistas. Dicha memoria es tratada

como un *buffer* circular, recorrido por un puntero. El tamaño del *buffer* marca cuánto retardo manifestará el sonido resultante, por ello es ajustable externamente. Para lograrlo, *fxMainControl* limita el máximo valor alcanzable por los punteros de lectura y escritura.

Todas las operaciones realizadas sobre las muestras, es decir, la atenuación y la mezcla, son simples. Por este motivo se efectúan combinacionalmente en el mismo ciclo de reloj. Para la mezcla se emplea un *audioMixerCombi*, el descrito en la sección 4.4.2. La atenuación consiste en dividir entre dos el dato, por medio de desplazar la muestra un bit a la derecha y después ajustar el signo, colocando un uno o un cero (según sea negativa o positiva respectivamente) en el bit más significativo.

El tamaño total de la *block ram* utilizada es de 32 KiB, lo cual permite hasta 0,168 segundos de *delay*. Se ha utilizado dicho tamaño debido a las restricciones que impone la FPGA.

Jamón

Este módulo requiere de una memoria, igual que el *delay*, porque pretende imitar un efecto Doppler (detallado en el capítulo anterior, apartado 3.1.7). Para ello es necesario recorrer al doble de velocidad el audio almacenado, provocando una alteración de tono que coincide con el segundo armónico, es decir una octava más agudo. Por tanto sonará "bien", en armonía con el tono original.

Como se observa en la figura 4.16 la arquitectura es muy similar al *delay*, no obstante los punteros para el acceso a memoria (*buffer* circular) tienen un relación distinta, ya que las escrituras y lecturas se realizan en direcciones distintas. Sin embargo, no se requiere que los accesos sean simultáneos, así que tampoco hace falta utilizar una *block ram* de doble puerto.

El componente principal *fxMainControl* gobierna el comportamiento del efecto mediante una máquina de estados. Al recibir una muestra de audio a procesar, pasa al estado de acceso a memoria, donde obtiene el dato retardado y almacena el recibido. También actualiza los punteros de acceso a memoria. A continuación procede a mezclar la muestra leída de memoria

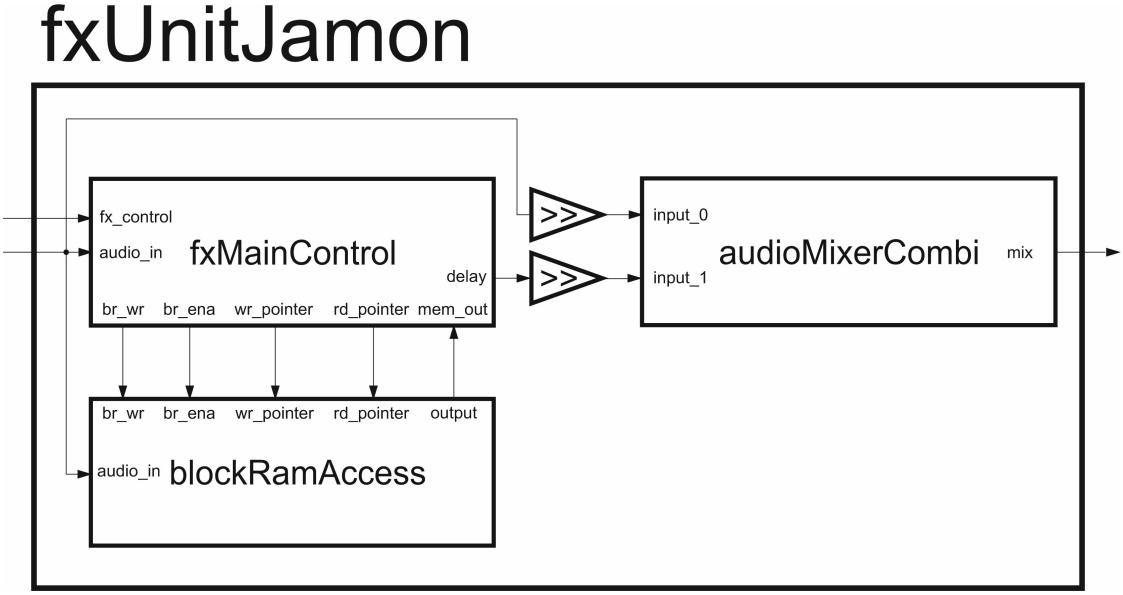


Figura 4.16: Arquitectura de *fxUnitJamon*.

y la recibida; el valor de ambas se divide combinacionalmente entre dos antes de la mezcla. Este último estado también notifica que el proceso ha terminado, ya que la operación no está registrada y su resultado queda directamente conectado a la salida.

El puntero de escritura avanza siempre de forma lineal y continua, cada muestra recibida se almacena en el *buffer* circular. Por otro lado, el de lectura avanza al doble de velocidad que el anterior, recorriendo todas las posiciones hacia delante hasta alcanzar al puntero de escritura. Llegado este punto, vuelve a recorrer en sentido contrario todas las direcciones, y así sucesivamente. En concreto, para realizar el avance a doble velocidad el puntero lee las muestras de dos en dos, es decir, se "salta" una; esta técnica es conocida como *downsampling* o *decimation*.

De forma similar al *delay*, el tamaño del *buffer* circular es ajustable en tiempo real, para variar el sonido resultante de aplicar el efecto.

Es necesario un *audioMixerCombi* para mezclar combinacionalmente la señal de entrada con la obtenida de la memoria. La reducción de volumen a la mitad de las muestras mezcladas se realiza con desplazamientos de un bit hacia la derecha.

El tamaño total de la *block ram* utilizada es de 16 KiB, lo cual provoca que el periodo

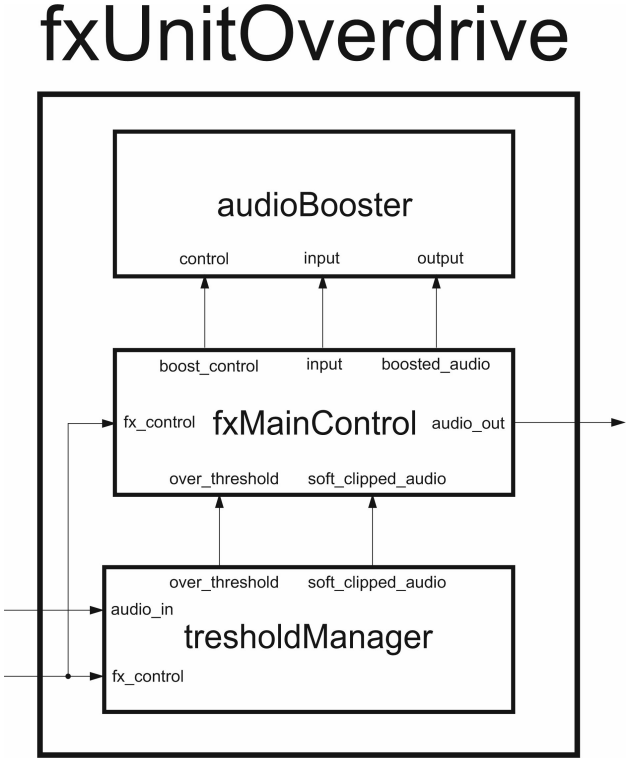


Figura 4.17: Arquitectura de *fxUnitOverdrive*.

de las oscilaciones en el sonido sea como máximo de 0,168 segundos. Se ha utilizado dicho tamaño debido a las restricciones que impone la FPGA.

Overdrive

Una de las características principales del *overdrive* es que la limitación de la onda no es brusca (*hard clipping*), como sucede con los mecanismos de control de saturación o el efecto de *fuzz*. En este efecto se implementa *soft clipping*, cuando se excede el umbral la onda se limita con suavidad. El sonido resultante es más "rico" y "cálido", puesto que mantiene cierta información de la onda original. El usuario puede ajustar en tiempo real el valor umbral

La arquitectura del efecto (figura 4.17) controla el valor que debe escribirse en el registro de salida mediante una máquina de estados finitos. En primer lugar dispone de un estado de espera, en el cual permanece hasta recibir una nueva muestra. En tal caso pasa al siguiente estado, donde se aplica el *soft clipping* al dato recibido, si la señal *over_ threshold* se encuen-

tra a alta, quiere decir que dicho dato está excediendo el umbral y debe ser reemplazado por el valor *soft_clipped_audio*. Ambas señales provienen del módulo *thresholdManager*, que limita el audio en función de el umbral (que puede ajustarse mediante *fx_control*). Para lograr la "suavidad" al limitar la onda, cuando la señal de entrada excede el valor máximo, la salida será el umbral más la muestra original multiplicada por un factor de reducción entre cero y uno. En este caso el factor de reducción empleado ha sido 0,25, pues equivale a dividir entre cuatro, y puede implementarse combinacionalmente desplazando la muestra. Recordando lo mencionado en el contexto tecnológico (sección 3.1.4), el comportamiento puede describirse con esta función:

$$softClipping(x) = \begin{cases} \frac{x-t}{4} + t, & x \geq t \\ x, & t \leq x \leq t \\ \frac{x-t}{4} + t, & x \leq -t \end{cases}$$

Donde x la muestra a procesar y t el valor umbral.

El problema principal que presenta limitar la señal es que su intensidad se ve reducida. Para paliarlo se utiliza un módulo *audioBooster* que amplifica las muestras, que realiza multiplicaciones por un factor superior a uno. De este modo se aumenta con precisión el volumen de salida, para equipararlo al de entrada. Para ello, en un nuevo estado somete la muestra a la acción del componente *audioBooster*, cuya ganancia (*boost_control*) será inversamente proporcional al valor umbral.

Compresor

Recordando lo mencionado en el capítulo anterior (apartado 2.1.4), a grandes rasgos este efecto reduce la amplitud de la señal cuando su volumen supera un cierto valor. Imita a su versión analógica, por lo que su acción no es inmediata y tiene un ataque moderado.

La arquitectura diseñada (figura 4.18) dispone de un mecanismo combinacional (*levelDetect*) para detectar si la señal supera el umbral de compresión, tal que el controlador principal (*fxMainControl*) pueda determinar la acción a emprender. Este último calcula un

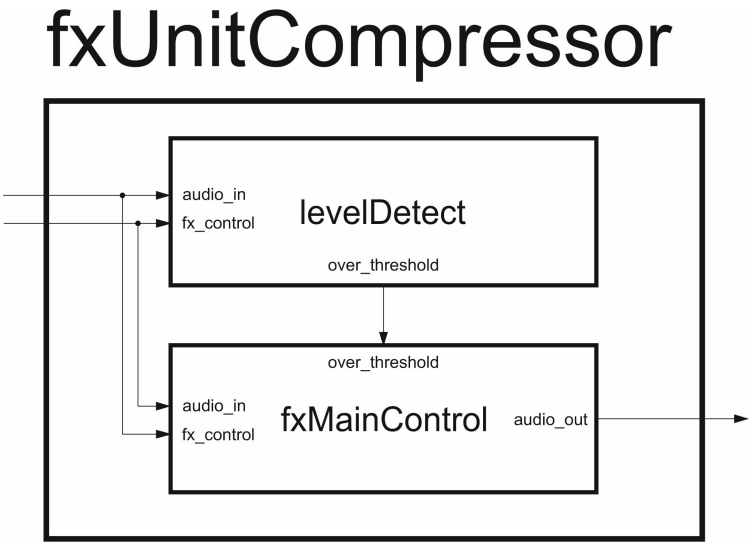


Figura 4.18: Arquitectura de *fxUnitCompresor*.

factor entre cero y uno, para reducir la amplitud de la muestra por debajo del valor límite.

Para regular el efecto, el valor umbral es ajustable por el usuario en tiempo real.

Internamente hay una máquina de estados finitos en el controlador principal que al igual que en los demás efectos, comienza su operación al recibir una muestra. En primer lugar multiplica (en punto fijo) la señal por la ganancia de compresión, que inicialmente es uno (es decir, no reduce nada). Un ciclo después, en el siguiente estado transmite al exterior el resultado. A continuación analiza dicho resultado para poder corregir (si fuese necesario) la ganancia de compresión, de este modo el tiempo empleado en el procesado de la muestra (desde que entra hasta que sale) es solo de tres ciclos.

El cálculo de dicha ganancia, por la cual se multiplica la muestra para atenuarla y así lograr el efecto de compresión, se realiza de forma similar a como se calculaba el llamado "factor de compresión" en el componente *CompressorMixer* (sección 4.4.2). Sin embargo la configuración es levemente distinta, pues la ventana es de 128 muestras (en lugar de 256) y el valor de cuenta para reducir la ganancia de compresión es 12 (en lugar de 50).

Esta ganancia se representa en punto fijo (Q2.6) y toma valores entre 1 y 0,19. Los incrementos y decrementos se realizan a intervalos de 0,015. Esto se debe a que es la máxima precisión que puede obtenerse empleando 6 bits para representar la parte decimal.

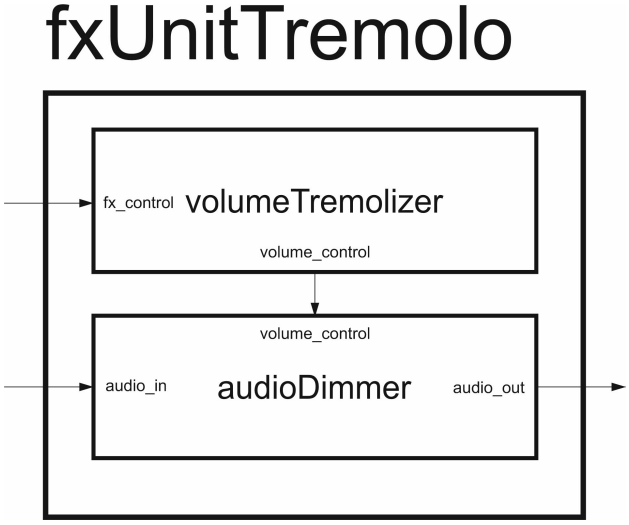


Figura 4.19: Arquitectura de *fxUnitTremolo*.

Comparativamente (con respecto a *CompressorMixer*) esto hace que el ataque sea más brusco, pues la ganancia de compresión puede variar en intervalos de 2,62 microsegundos (la mitad). También se reduce al 10 % el porcentaje de muestras que deben exceder el umbral para antes de decrementar la ganancia. Esto resulta en una acción del compresor más agresiva y audible.

Los motivos de ello son por un lado, que las pistas en general trabajan con muestras de menor amplitud que el mezclador de salida, puesto que en este último se trabaja con todas las pistas mezcladas en una señal. Al ser menor la amplitud también es más difícil que las muestras alcancen el umbral.

Por otra parte, en el procesador de efectos se pretende marcar con mayor intensidad la presencia del compresor (que se aprecie claramente), mientras que en el mezclador de salida se utiliza como un mecanismo para prevenir la saturación y el exceso de volumen y por ello el objetivo es que pase desapercibido en medida de lo posible.

Trémolo

El trémolo es un efecto relativamente simple. El funcionamiento (sección 2.1.4) se resume en variar cíclicamente el volumen de salida.

Como se observa en el diagrama 4.19, su implementación emplea un módulo *audioDimmer* para ajustar la amplitud, mediante multiplicaciones en punto fijo (Q2.4) y una máquina de estados para controlar la variación del volumen.

Dicha máquina de dos estados regula la frecuencia a la que varía el volumen empleando un contador, cuyo máximo valor es ajustable (*fx_control*). Al vencer la cuenta se incrementa o decrementa (según corresponda) el factor de volumen. El periodo de oscilación se ha establecido entre 0,08 y 1,34 segundos, un rango bastante amplio y habitual en este tipo de efecto.

Asimismo, cuando llega al módulo una nueva muestra, es multiplicada por dicho factor de volumen y se transmite al exterior. El valor mínimo factor de volumen se ha establecido en 0,2 en lugar de 0 para que la atenuación no sea demasiado brusca.

Fuzz

Este efecto es un tipo de distorsión, más agresiva que el *overdrive*, la principal diferencia es que implementa *hard clipping* asimétrico. De esta manera no se suaviza la parte de la onda que queda limitada; además existen dos valores umbral distintos, uno para las amplitudes positivas (es decir, mayores que cero) y otro para las negativas. También provoca la aparición de armónicos pares en la onda resultante.

La arquitectura, como muestra la figura 4.20 consiste en un módulo amplificador (*audioBooster*), lógica combinacional para aplicar la limitación asimétrica y un atenuador (*audioDimmer*).

FxMainControl, al igual que en otros efectos, contiene una máquina de estados que regula su comportamiento. Cuando una muestra entra en el componente, es amplificada (en función del parámetro que regula la intensidad del efecto), mediante un multiplicador. Así se fuerza a la onda a sobrepasar los umbrales. En el siguiente estado se aplica la limitación asimétrica mediante una simple operación combinacional, cuya descripción funcional atiende a lo siguiente:

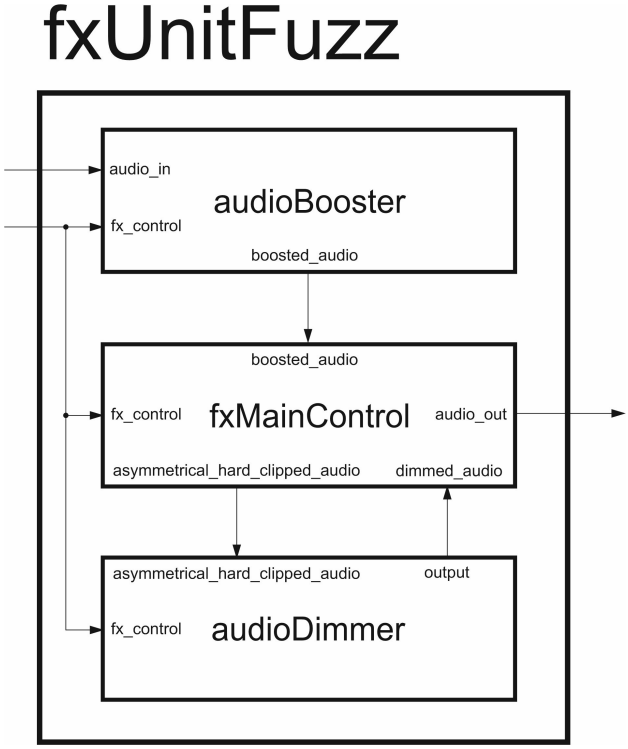


Figura 4.20: Arquitectura de *fxUnitFuzz*.

$$hardClipping(muestra) = \begin{cases} muestra, & \text{si } muestra < 0 \\ muestra, & \text{si } muestra \geq 0 \text{ y } muestra < límite \\ límite, & \text{si no} \end{cases}$$

En el proceso se va a ver aumentado el volumen de salida, y es necesario corregirlo mediante el módulo atenuador. Para ello, en el estado que sigue se aplica la operación, almacenando el resultado en el registro de salida.

Cuando el usuario ajusta el parámetro de intensidad en este efecto, actúa directamente sobre la ganancia del *audioBooster*.

El umbral (es decir, el limite) es fijo, y se ha establecido en el máximo valor representable con 25 bits (en C2). Las muestras se codifican con 32 bits, aunque el audio códec las proporciona de 24 bits; por tanto si la ganancia se mantiene en su valor mínimo (es decir, uno), las muestras no se verán saturadas porque tendrán un valor inferior al umbral. Al

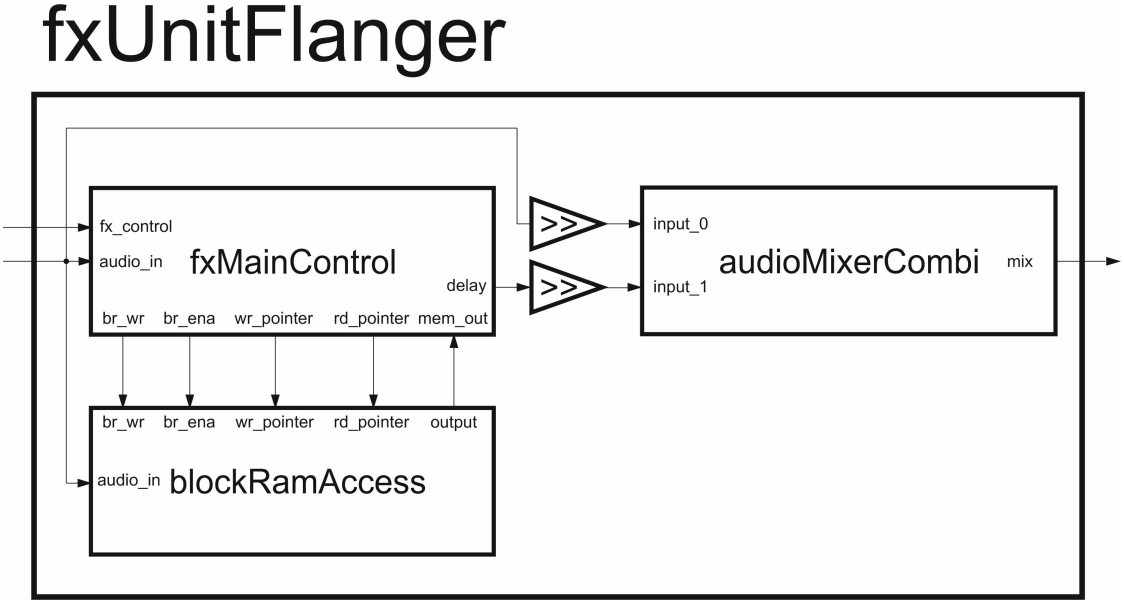


Figura 4.21: Arquitectura de *fxUnitFlanger*.

incrementar la ganancia empezará progresivamente a apreciarse el efecto en el sonido.

El valor de atenuación en el componente *audioDimmer* también depende del ajuste del usuario, tal que la amplificación quede compensada y que el volumen de salida no se vea incrementado en desmedida.

Flanger

El *flanger* es un efecto de sonido oscilante, basado en mezclar la señal original con otra de retardo variable.

Arquitectónicamente es idéntico al jamón, como refleja la figura 4.21. Sin embargo internamente implementa un contador para lograr la oscilación de baja frecuencia, necesaria para variar periódicamente el tiempo de retardo de las muestras. Además el tamaño del *buffer* es también mucho menor, porque el retardo en este efecto no debe exceder los 2 milisegundos. El máximo tiempo de *delay* del efecto es ajustable en tiempo real, tal que el usuario pueda configurar a qué velocidad oscila el efecto (y su profundidad).

La máquina de estados principal dispone de un estado inicial que establece el tiempo de retardo de acuerdo al contador. Cuando es recibida una nueva muestra, pasa al siguiente

estado, que destina un ciclo a actualizar los punteros de lectura y escritura, así como a almacenar dicha muestra en memoria (no necesita más, pues se trata de *block ram*). A continuación se destina otro estado de un ciclo de duración a completar la mezcla de la señal original y la retardada. Finalmente se notifica al exterior que la operación ha concluido.

La mezcla se realiza con un *audioMixerCombi*, cuyas entradas son combinacionalmente divididas entre dos, por medio de un desplazamiento a la derecha con control de signo.

El tamaño total de la *block ram* utilizada es de 512 bytes; de este modo se obtienen como máximo 2,62 milisegundos de retardo. Recordando los detalles del *flanger* este debe funcionar con *delays* de entre 0 y 2 milisegundos.

HiPass y *LoPass*

Estos efectos son esencialmente un filtro paso alta y paso baja, sirven para eliminar de la señal el exceso de graves o agudos respectivamente. El filtro que emplean internamente es de tipo FIR (*Finite Impulse Response*). La decisión de utilizar este frente a uno IIR se debe a la ausencia de realimentación, que lo hace más estable y los posibles errores que puedan producirse desaparecen totalmente después de tantas muestras como etapas tenga el filtro.

Disponen de múltiples frecuencias de corte, seleccionables en tiempo real. Para el filtro paso baja se han escogido (en Hz) 15000, 3500, 1000, 400 y 220; mientras que para el paso alta han sido 25, 40, 80, 140 y 190. Los valores anteriores se han obtenido de implementaciones comerciales de estos filtros (en concreto los mostrados en el capítulo previo dedicado a la funcionalidad, figura 2.11).

Tras una serie de simulaciones y pruebas se estimó lo más conveniente fijar el número de etapas (es decir, de muestras almacenadas en RAM y coeficientes) en 511. Con menos, la precisión de los filtros se ve comprometida.

La arquitectura de los mismos utiliza ROMs con constantes almacenadas, una RAM que guarda un número predefinido de muestras y un MAC (*Multiply-Accumulate*: multiplica dos valores y acumula el resultado, sumándolo al anterior). Se observa un esquema general en la figura 4.22.

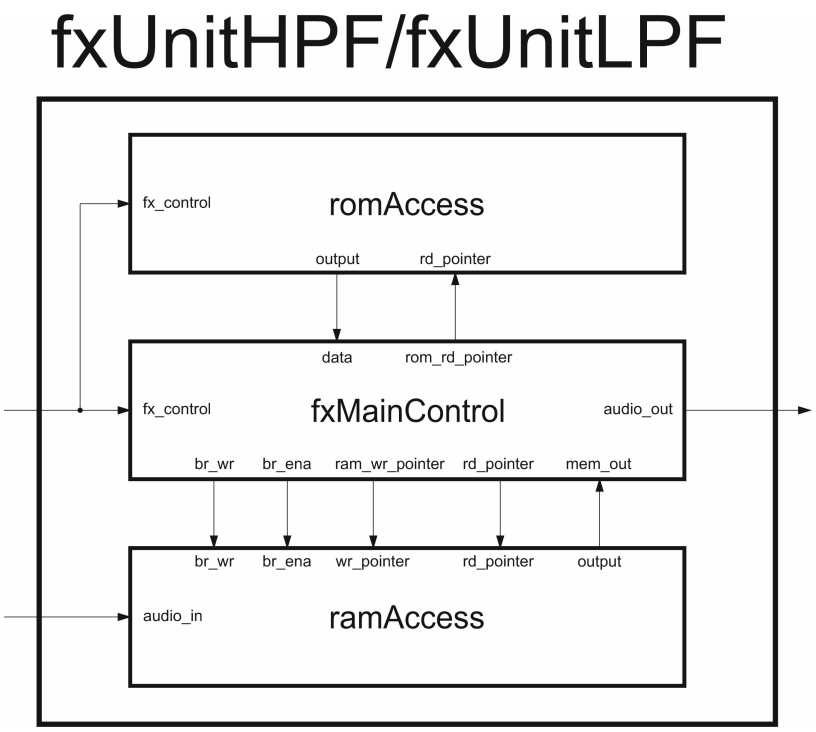


Figura 4.22: Arquitectura de *fxUnitHPF* y *fxUnitLPF*.

El componente *ramAccess* gestiona el acceso a la memoria *block ram*, que actúa como *buffer*, ya que almacena las últimas 511 muestras recibidas. Por otro lado, *romAccess* administra el acceso a las ROMs que contienen las constantes necesarias para filtrar las señales. Como hay varias frecuencias de corte son necesarias varias ROMs, por lo que *romAccess* también se encarga de seleccionar la apropiada.

Para el cálculo de las constantes se ha utilizado el algoritmo de la ventana de Hamming. Entre otras ventajas de este, las constantes que genera son simétricas; en otras palabras, las primera mitad de la tabla de constantes coincide con la segunda. Por ello es suficiente con una ROM de la mitad de profundidad (en este caso, 255 posiciones), aunque esto también complica levemente la administración del puntero de acceso a la misma.

El algoritmo de filtrado lo gestiona la FSM de *fxMainControl*. Se trata de diseño multiciclo que sucesivamente realiza las multiplicaciones y sumas necesarias para hallar el valor acumulado, empleando un ciclo por etapa, en este caso 511.

En primer lugar cuando llega un nuevo dato, el sistema sale del estado de espera para

almacenar dicha muestra en la RAM y avanzar el puntero de escritura. Al ciclo siguiente empieza a filtrar, para ello multiplica una a una las muestras de audio del *buffer* por los coeficientes almacenados en ROM y suma todos los resultados. Cada una de estas operaciones (multiplicación con acumulación) se realiza en un ciclo.

Simultáneamente se actualizan los punteros de acceso a memoria: en el caso de las muestras consiste en sumar uno cada vez (salvando el momento en que "da la vuelta" al *buffer* circular). Para los coeficientes hay que considerar la propiedad simétrica de estos y recorrer la ROM desde la dirección más baja a la más alta durante los primeros 255 ciclos y luego al contrario (desde la más alta a la más baja).

Este efecto es probablemente el más costoso (en términos de tiempo de cómputo) de todos las presentes en el flujo del audio, puesto que cada multiplicación necesita un ciclo de reloj.

4.8. Control

En esta sección se cubren los componentes destinados al control del sistema por parte del usuario, de ellos proceden las señales de control (ya introducidas en la sección 4.2 para describir la comunicación entre módulos).

Como breve recordatorio, el control consiste principalmente en señales que transmiten pulsos para indicar qué acciones debe tomar el sistema. El principal objetivo que persiguen estos módulos es traducir la información recibida desde el teclado y el *Bluetooth* (controlados por el usuario), para que el resto de componentes (pistas, efectos...) puedan interpretarla.

Se ha diseñado de esta manera para evitar tener que almacenar toda la información del sistema en el componente de control. Dicha alternativa eliminaría la necesidad de tener que enviar los pulsos a los componentes, pero complica considerablemente la detección de cambios de estado y dificulta mantener la coherencia, pues las actualizaciones no serían inmediatas. Por ello, cada módulo gestiona su propio estado de forma individual e independiente.

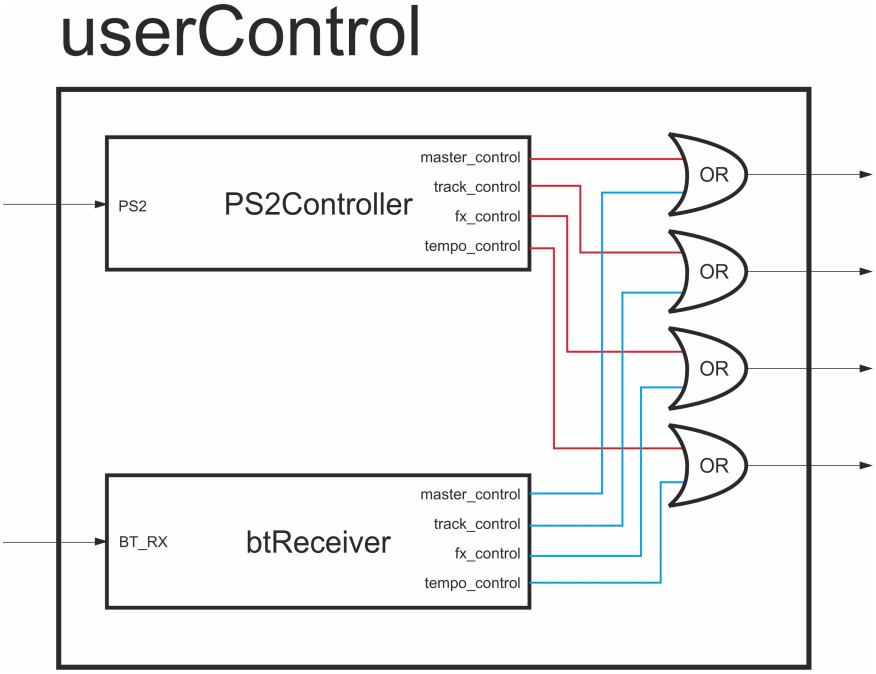


Figura 4.23: Componentes de control unificados.

4.8.1. *UserControl*

UserControl es el módulo principal de control. Unifica las distintas fuentes presentes para el manejo del sistema, tal que sea posible utilizar todas al mismo tiempo.

Contiene los componentes *PS2Controller* y *btReceiver*, que reciben tramas desde el teclado y el receptor *Bluetooth* respectivamente. Ambos generan pulsos de control en base a los datos recibidos.

4.8.2. Control *Bluetooth* (*btReceiver*)

Este componente recibe y parsea los datos recibidos por *Bluetooth*, para generar las señales de control correspondientes.

Las tramas que provienen de la comunicación se reciben en serie (mediante RS-232 con ocho bits de datos y uno de *stop*, a 115200 baudios por segundo [14]). Esto se debe la implementación del dispositivo empleado. El módulo *rs232Receiver* construye palabras de tamaño *byte* a partir de los datos recibidos [26].

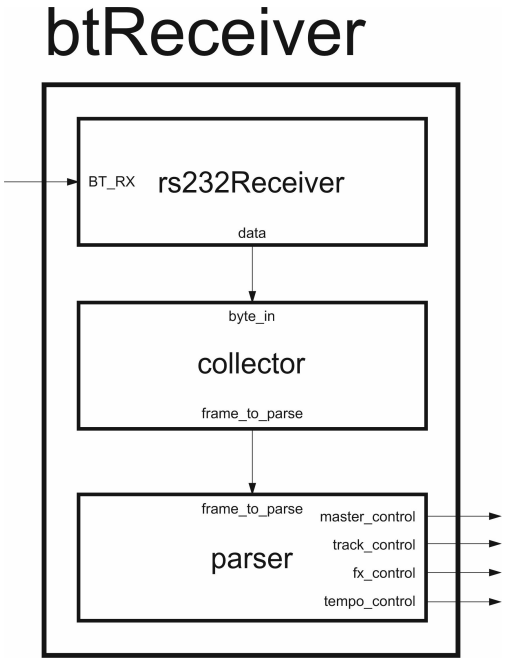


Figura 4.24: Receptor de tramas *Bluetooth*.

Para minimizar el impacto de la comunicación en el ancho de banda, así como en el consumo, el controlador *Bluetooth* solo envía información cuando se produce una interacción en el mismo. Es importante remarcar que el controlador en ningún caso almacena o mantiene el estado de ningún dato, solo transmite la acción a realizar sobre el sistema. Gracias a ello se evitan problemas de consistencia.

Las tramas se montan en *collector*, que está implementado mediante una FSM. En el estado inicial, espera la recepción de *bytes* desde *rs232Receiver*. Cuando recibe tres (que hacen los 24 bits necesarios), envía el resultado a *parser*. Si el contenido del primer *byte* recibido coincide con "00100101" (llamado *metadata_ byte*), pasarán a descartarse todos los datos entrantes, es decir, se deshabilitará la construcción de tramas. Para salir de este estado es preciso volver a recibir otro *metadata_ byte*. Esto sirve para la recepción de información ajena al control del sistema, así como para descartar metadatos generados por el controlador de *Bluetooth* utilizado. Este sistema se implementa utilizando una máquina de estados.

Finalmente, las tramas montadas se parsean combinacionalmente en el módulo *parser*. También genera el pulso en la línea de control correspondiente. Para los casos que procede

(por ejemplo, volumen, compresión *overdub*...), también escribe el valor recibido en la señal asignada.

Trama de control

El formato de trama es el mostrado en la figura 4.25. La longitud es fija: 24 bits.

En primer lugar, los cinco bits más significativos (23-19) indican el modo. Este permite clasificar el mensaje en función de a qué módulo del sistema está destinado. Hay valores especiales del campo "modo": *reset* (para efectuar un *reset hardware* en el *looper*) y *forbidden* (para prevenir que las tramas de comunicación coincidan con el antes citado *metadata_ byte*).

Los valores de los campos que ocupan los bits más significativos condicionan la forma en que se interpretan los demás. Por tanto, en función del modo, la información codificada en los campos pista, opción y valor cobrarán un significado distinto.

Los bits del 18 al 16 codifican a qué pista afecta el mensaje. Dado que hay ocho pistas se ha fijado su longitud en tres bits. Cuando el mensaje es de modo "master" este campo se utiliza para indicar a qué canal de salida se debe aplicar la acción contenida en el mensaje.

A continuación, el campo "opción" (de un byte de anchura) sirve para manifestar a qué parámetro afecta el mensaje, dentro del modo en cuestión. En caso de que el modo sea "master", este campo puede tomar los valores "compresión" o "volumen". Cuando el modo es "track", opción puede ser "reproducir", "grabar", "*mute*", "volumen"... entre otros. Si el modo es "fx", las opciones son "encolar", "limpiar cola", "activar", etcétera.

Por último, el campo "valor" (también de un byte de anchura) completa la información sobre la acción de control, especificando la cuantía de la misma. No se aplica en todos los casos, por ejemplo cuando el campo modo es "track" y el campo opción es "grabar" o "reproducir". Sin embargo, cuando se pretende enviar un mensaje para establecer el volumen de una pista o la intensidad de un efecto, sí es necesario indicar el valor asociado utilizando este campo.

Cuando se envía un mensaje de control relativo a los efectos, el campo valor se utiliza para especificar el identificador del efecto al que va dirigido. Esto se ha especificado así con

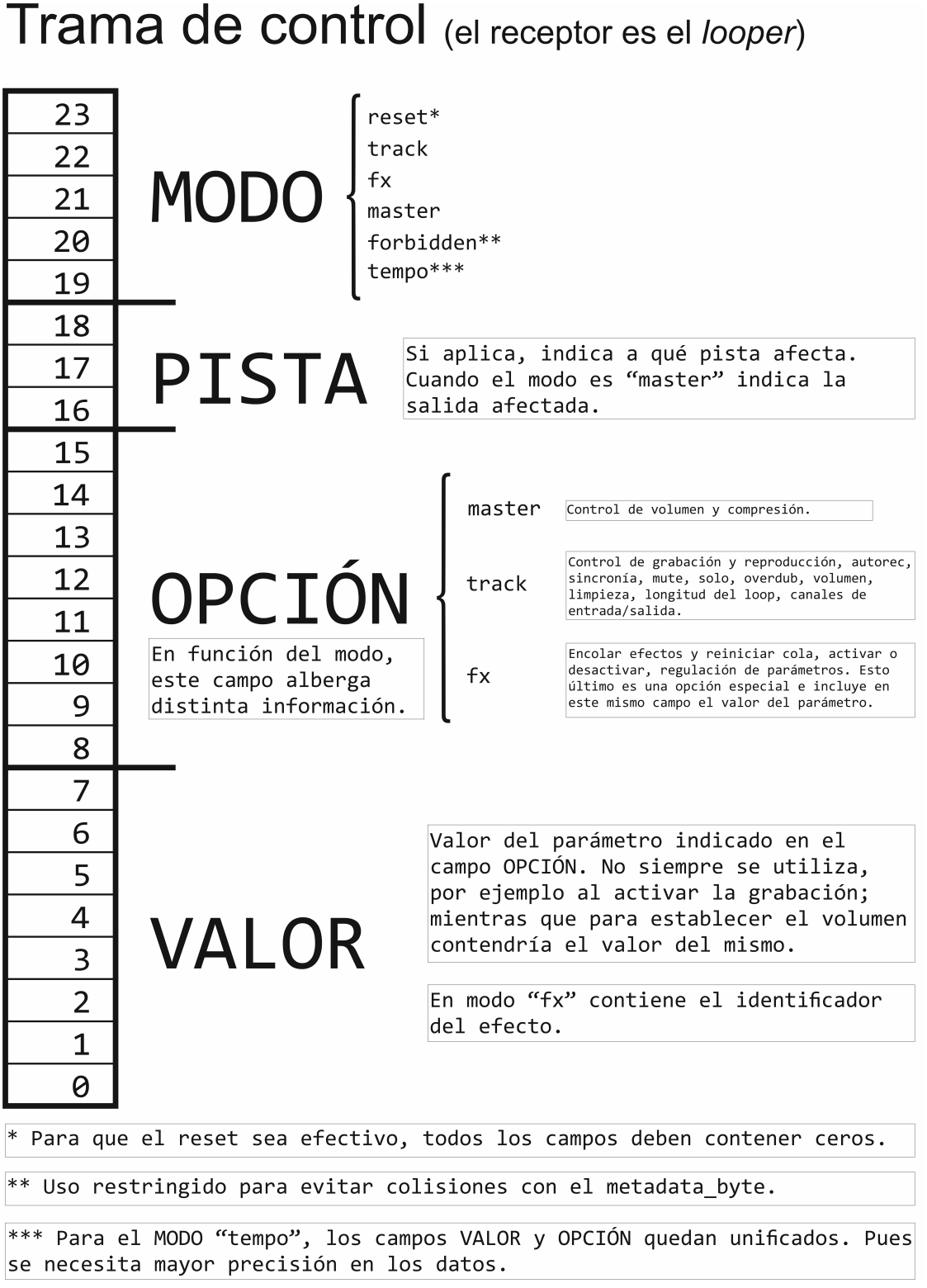


Figura 4.25: Trama de control.

vistas a poder controlar hasta 256 efectos distintos por pista sin necesidad de modificar el formato de trama (puesto que el campo valor se codifica con ocho bits).

Hay una situación especial en el uso de estos campos, sucede cuando el modo del mensaje es "tempo". En este caso los campos pista, opción y valor se utilizan solapados, para enviar el *tempo* con mayor precisión. Así se dispone de 19 bits para codificarlo (frente a los ocho que tienen los campos valor y opción).

4.8.3. Control por teclado (*PS2Controller*)

Para implementar el control por teclado este módulo incluye el componente *ps2KeyboardInterface*, que devuelve los códigos asociados a las teclas. En él está implementado el protocolo PS/2: recibe los datos en serie y realiza una comprobación de paridad. El diseño de este módulo es una adaptación de material docente de la asignatura optativa *Diseño Automático de Sistemas* (impartida en el grado al que corresponde este trabajo) [12, 27].

El componente *keyScanner* consiste en una máquina de estados que gestiona los eventos de pulsación y liberación de las teclas, generando de forma combinacional los pulsos de control que corresponden. Dicha máquina permanece en un estado inicial hasta recibir un evento desde *ps2KeyboardInterface*. Si el evento recibido es de pulsación genera el pulso de control asociado a la tecla; si por el contrario el evento corresponde a una despulsación, pasa a otro estado para esperar a recibir el código de la tecla despulsada (aunque no tiene ningún efecto de cara al control del sistema).

Puesto que el número de teclas del que dispone un teclado es limitado, no es posible ni por asomo controlar todos los parámetros de las pistas. Por ello se ha solventado la situación implementando un seleccionador de pistas (*selecCounter*); tal que la pista que se encuentre seleccionada, será aquella sujeta al control de ciertas teclas "compartidas". Por ejemplo, para regular el volumen con el teclado, es preciso primero seleccionar la pista en cuestión y después realizar el ajuste.

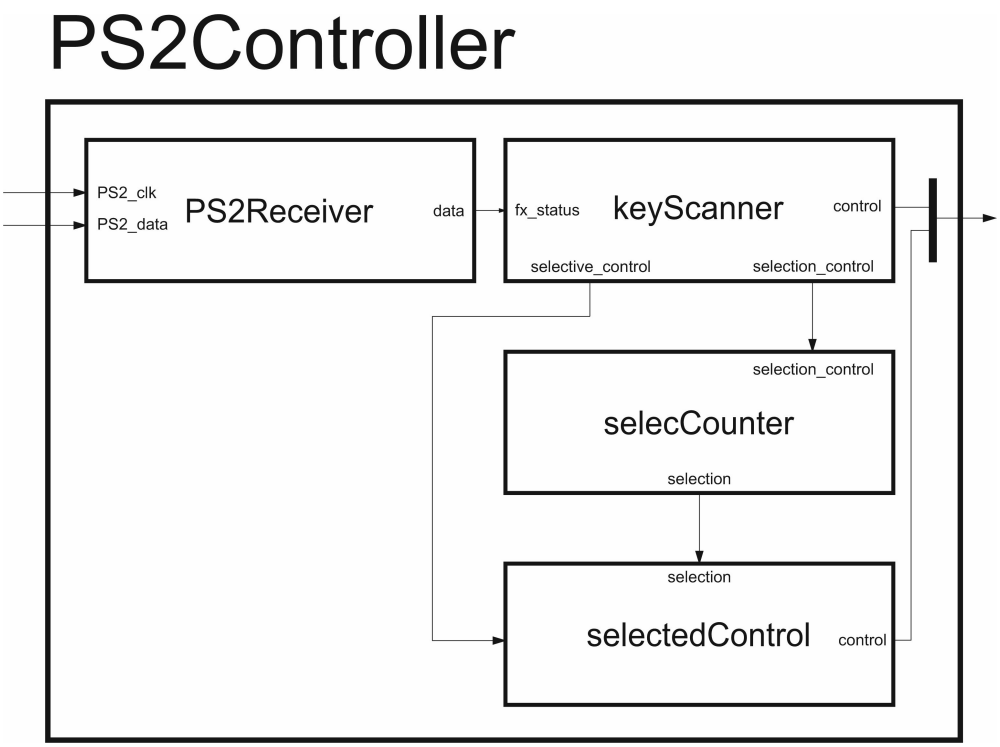


Figura 4.26: Controlador del teclado.

En general, las teclas alfanuméricas centrales están orientadas al control de las pistas. Cada fila está asociada a una función, tal que sendos números (del uno al ocho) controlan la reproducción de las pistas; ídem para el mute, la grabación, y el borrado. Con las teclas "ñ", "p", "[" y "{" (distribución QWERTY en España) se permite ajustar el volumen ambas salidas.

El resto de funciones no disponen de una tecla directa, la selección de pista se realiza con el cero y el punto del teclado numérico. Las teclas de función permiten ajustar los canales de entrada, el volumen, la sincronización y el solo.

Por último, el teclado numérico permite controlar el encolado y activación de los efectos, así como el ajuste de sus parámetros.

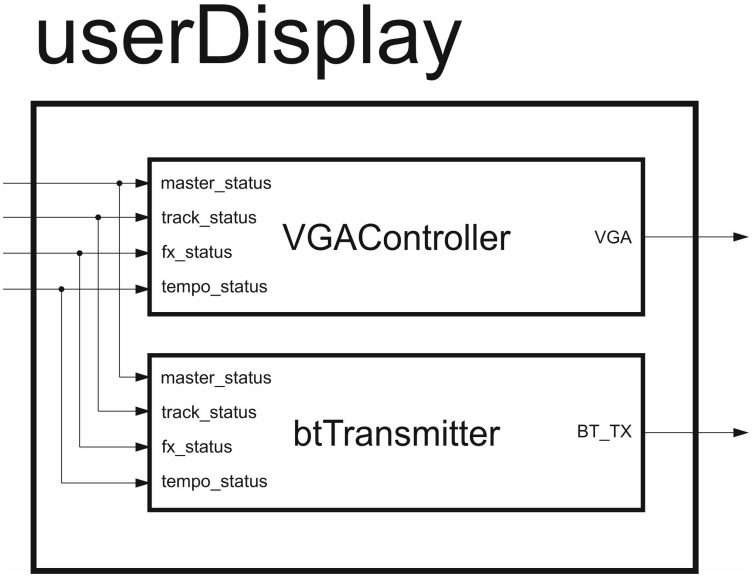


Figura 4.27: Envoltorio de *btTransmitter* y *vgaController*.

4.9. Visualización

Para que el usuario pueda visualizar en tiempo real el estado del *looper*, son necesarios los componentes *btTransmitter* y *vgaController*. Ambos se sirven de las mismas señales para transmitir la información del sistema, sin embargo funcionan de forma independiente y paralela. No dependen uno de otro, de hecho sus arquitecturas son radicalmente distintas.

Las señales de estado (como se describe en la sección 4.2) proceden de los distintos módulos (pistas, efectos, etcétera). En general corresponden a la salida de registros y van acompañados de otra señal que notifica las actualizaciones mediante un pulso.

4.9.1. *UserDisplay*

Este módulo simplemente sirve para envolver al *btTransmitter* y *vgaController* antes mencionados.

En la figura 4.27 se puede comprobar que no contiene ningún tipo de lógica, solo conecta los componentes, sin embargo está presente para respetar la jerarquía del diseño.

4.9.2. Envío del estado por *Bluetooth* (*btTransmitter*)

A grandes rasgos, para enviar la información por *Bluetooth*, el sistema detecta cuando se han producido actualizaciones en los datos, los serializa y los pasa al módulo externo que realiza la comunicación. De esta manera solo se envían datos en momentos puntuales (al igual que en la recepción, sección 4.8.2), optimizando el consumo y uso de la red.

Respecto a otras alternativas de implementación considerablemente más sencillas, por ejemplo enviar periódicamente toda la información de estado, el diseño elaborado mejora el tiempo de respuesta. Se debe a que no es necesario limitar el refresco a un cierto periodo. Puesto que dicho factor es crucial en un sistema de tiempo real, se ha decidido optar por este diseño enfocado a enviar las actualizaciones de estado lo antes posible.

No es trivial la arquitectura necesaria para ello, puesto que múltiples cambios pueden tener lugar simultáneamente y no es inmediato construir la trama ni enviarla en serie. La solución diseñada (figura 4.28) se compone de unos registros que almacenan qué datos han sido actualizados (*flagSet*), los cuales son leídos por (*frameBuilder*) para elaborar las tramas.

Este último componente emplea una máquina de estados que iterativamente comprueba conjuntos de *flags* (mediante un puntero de posición), clasificados en trece grupos. Cuando uno de los conjuntos está activo, pasa a un estado que nuevamente itera sobre los *flags* del mismo y crea tramas para aquellos que lo requieran. Después de la operación, los *flags* son limpiados y las tramas almacenadas en un registro. A continuación, en un nuevo estado, el contenido de dicho registro es enviado a la fifo (*sendFifo*). Acto seguido vuelve al estado principal, para seguir iterando.

Para los efectos se complica ligeramente el mecanismo, dado que por cada pista hay varios efectos. Esto exige una iteración anidada sobre los *flags* que engloban todos los efectos de cada pista; y cuando uno de ellos esté activo, otra iteración sobre los *flags* asociados a cada efecto.

El sentido de agrupar los *flags* es por un lado acelerar el proceso al no tener que analizar todos constantemente (conllevaría demasiados ciclos) y por otro reducir los recursos *hard-*

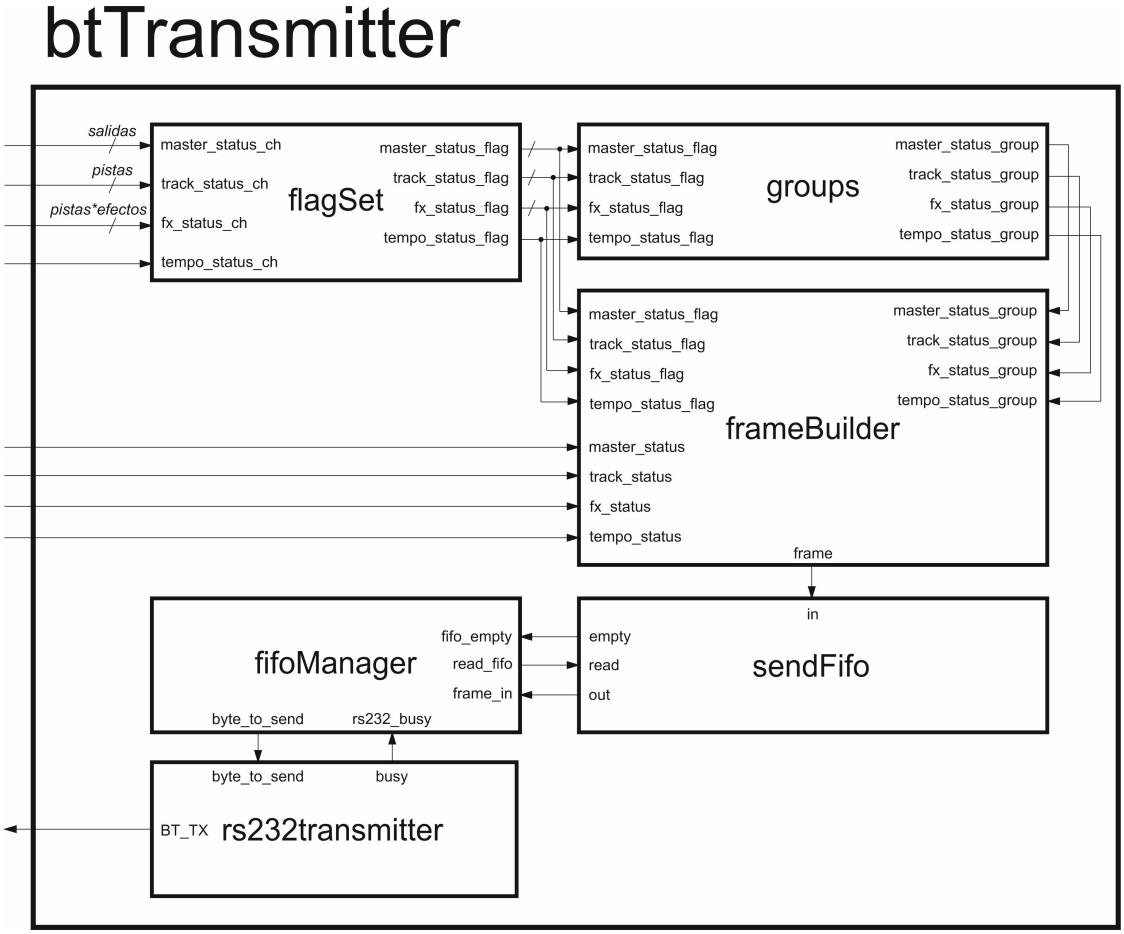


Figura 4.28: Generador de los mensajes a transmitir por *Bluetooth*.

ware (y el consumo dinámico) al evitar inmensos multiplexores y varios niveles de lógica combinacional.

El contenido de las tramas se genera combinacionalmente; por un lado, en función del estado de las iteraciones se obtiene el grupo actual (volumen, longitud del *loop*, *overdub*...), la pista o canal de entrada o salida a los que afecta, el efecto (si aplica), etcétera. Por otra parte el valor asociado al elemento cuyos *flags* están activos, se obtiene desde las señales de estado que recibe este módulo.

El componente *fifoManager* se ocupa de coordinar la fifo con el serializador de tramas (*rs232Transmitter*). Si la fifo contiene datos y el serializador no está funcionando, pasa las tramas *byte* a *byte* a este último. Como es habitual, se implementa con una máquina de estados (tres en este caso).

Los datos que se envían en serie al módulo *Bluetooth* respetan el formato indicado en su manual, es decir, ocho bits de datos y uno de *stop*, sin paridad. [14]

Además de controlar el *looper* es posible enviar peticiones para efectuar un *reset hardware* al sistema.

La figura 4.29 muestra un ejemplo del *LoopMAN* siendo controlado a través de *Bluetooth*.

Trama de visualización

Respecto a las tramas, son de 24 bits (longitud fija). Como se muestra en la figura 4.30, son muy similares a las empleadas en el envío de estado, sin embargo presentan sutiles diferencias para optimizar la comunicación, condensando cierta información (por ejemplo, el estado de las pistas) para reducir el número de tramas enviadas. Así también se simplifica el diseño del *frameBuilder*.

Los campos son los mismos que los ya introducidos en la trama de control (sección 4.8.2): modo, pista, opción y valor.

Aunque los mensajes que transportan son esencialmente los mismos, hay ciertos detalles a considerar, teniendo en cuenta que ahora el emisor es el *looper*. Por ejemplo, cuando el campo "modo" contiene el valor "reset", en lugar de una petición de *reset hardware* al *looper*,

Figura 4.29: *LoopMAN* controlado remotamente.

lo que se está manifestando es que se ha efectuado dicha acción en el sistema.

El campo "pista" funciona igual que en la trama de control.

Por el contrario, "opción" cambia para los modos "track" y "fx". En el primer caso, puesto que ahora no se trata de enviar mensajes puntuales de control, sino de retransmitir el estado del sistema, existe una opción para enviar en un solo mensaje todos aquellos parámetros de las pistas que son representables con solo un bit. Esto incluye el estado de la reproducción, si está grabando, si está activado el *mute*, etcétera. La razón por la cual no es posible hacer esta optimización en las tramas de control que recibe el *looper*, es que ello exigiría mantener en el controlador remoto el estado del sistema para poder enviar la información actual de cada parámetro. Sin embargo, el objetivo del control remoto se limita a transmitir simples eventos informando de la pulsación de un botón o un cambio en el valor de un *slider*, puesto que se trata de un terminal y no debe en ningún caso mantener el estado ni implementar lógica. Esto atiende a motivos de seguridad y coherencia de memoria.

Al enviar la información de las pistas que sí tiene un valor asociado, el formato es idéntico

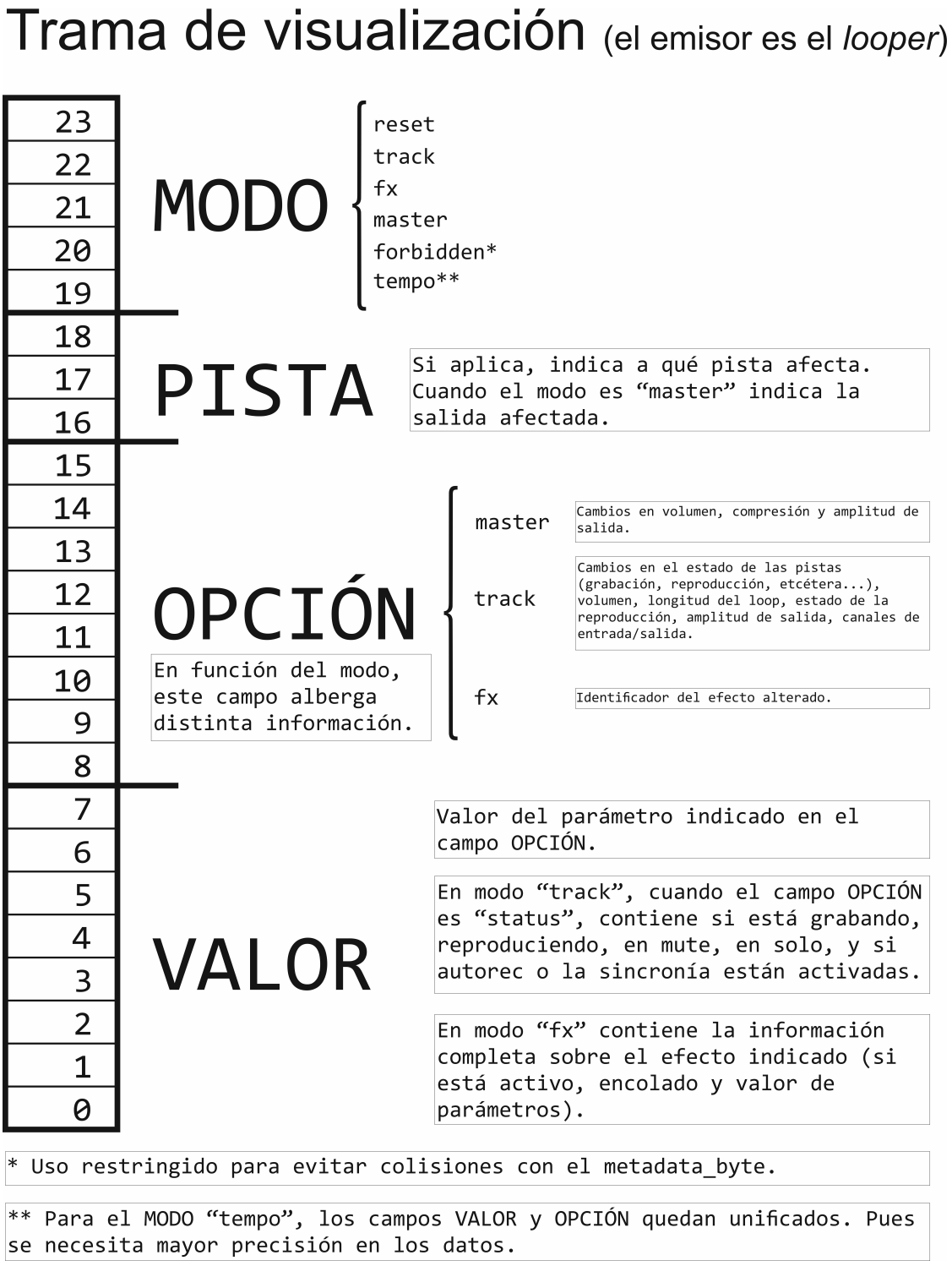


Figura 4.30: Trama de visualización.

a la recepción de mensajes. Es el caso del volumen, la longitud de la grabación o la amplitud actual de la señal (evidentemente este último no existe para las tramas de control, ya que el sonido se genera en el *looper* y no en el *tablet*).

Cuando el modo es "fx", el campo "opción" alberga el código de ocho bits que identifica al efecto cuyo estado está siendo transmitido. En el campo "valor" se incluye toda la información relativa a dicho efecto: si está encolado, si está activo y el valor de su parámetro ajustable.

4.9.3. Visualización del estado en monitor VGA (*vgaController*)

VgaController toma las señales de estado para generar los gráficos combinacionalmente.

El componente *vgaInterface* envía la señal de vídeo al exterior del sistema, de acuerdo al protocolo VGA, tal y como fue descrito en la sección 3.2.3. Para ello barre todos los píxeles de la pantalla: mediante las señales *pixel* y *line* notifica el píxel actual, y por cada uno de ellos recibe a través de la señal *RGB* el color asociado al mismo. El diseño de este módulo es una adaptación de material docente de la asignatura optativa *Diseño Automático de Sistemas* (impartida en el grado al que corresponde este trabajo).

De este modo, *vgaTrack*, *vgaMaster*, *romGraphics* y *tempoVisualizer* manifiestan el color del píxel que *vgaInterface* solicita. Las señales de color de todos ellos se unifican mediante puertas OR.

Los ocho *vgaTrack* representan las pistas y su estado (volumen, *beat*, si está grabando, reproduciendo, en *mute*...). Hay dos *vgaMaster* para dibujar las salidas de audio. Por otro lado el módulo *romGraphics* gestiona el acceso a unas memorias ROM que almacenan los números de las pistas y el logo de *LoopMAN*. En último lugar, *tempoVisualizer* contiene una máquina de estados temporizada que se encarga de mostrar el *tempo*; implementa un mecanismo para mantener visible durante un tiempo un indicador, que se activa al recibir un pulso en la señal *tempo*. Dicho comportamiento se logra con una FSM temporizada de dos estados, uno de espera y otro de cuenta regresiva.

La figura 4.32 muestra el sistema en funcionamiento, conectado a un monitor VGA. En

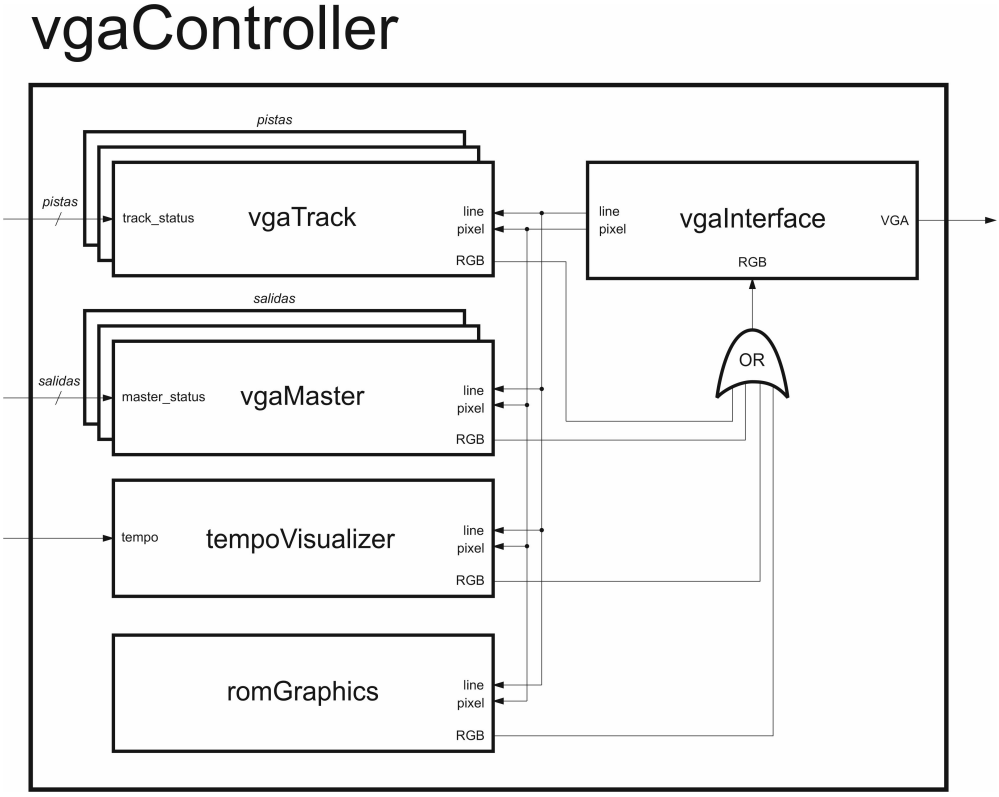


Figura 4.31: Controlador VGA.

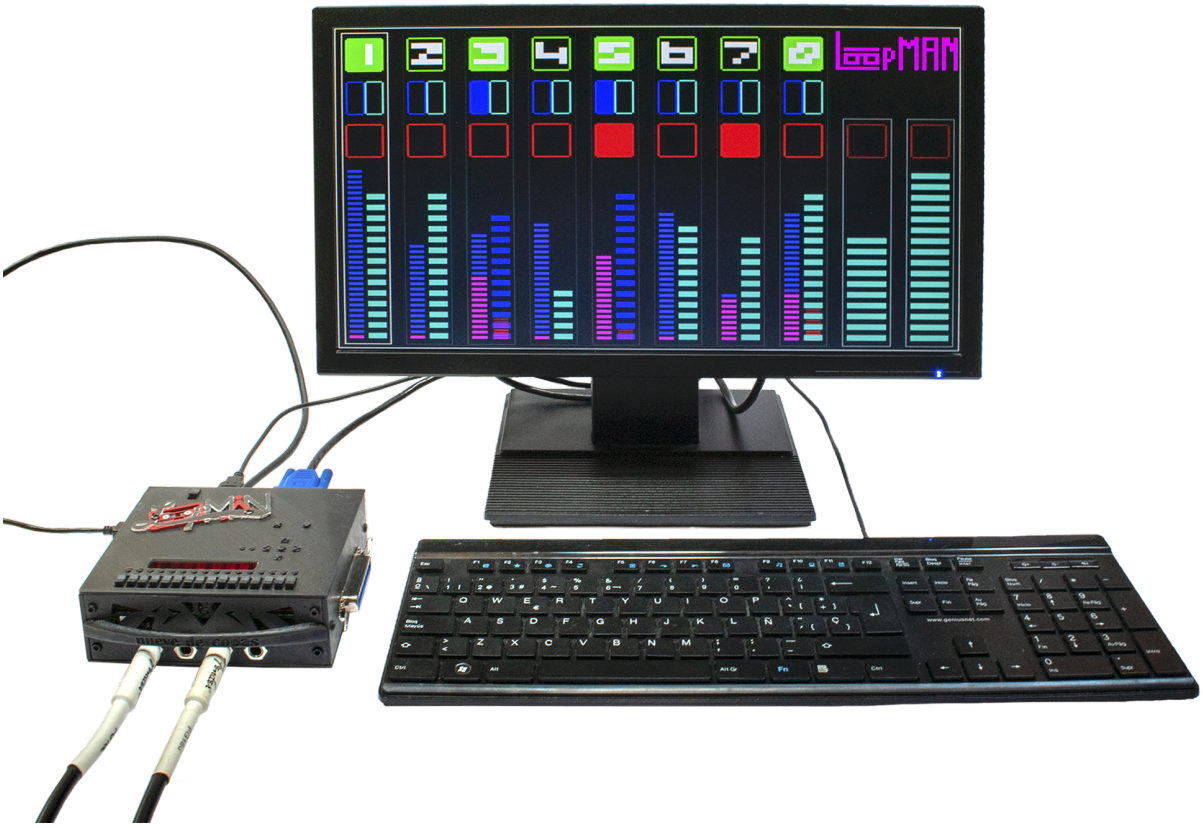


Figura 4.32: *LoopMAN* conectado a un teclado y monitor VGA.

la interfaz gráfica dibujada por el *looper* se observan las ocho pistas (numeradas), dispuestas horizontalmente. Cada una de las cuales incluye, de arriba a abajo, el estado de la reproducción, *mute*, solo y grabación. A continuación, en forma de barras muestra la longitud del *loop* grabado y el pulso actual (a la izquierda); y el volumen e intensidad del sonido (a la derecha).

Debajo del logo de *LoopMAN*, se encuentra el metrónomo. Puesto que parpadea para indicar los pulsos, en la imagen no se aprecia su utilidad.

Por último se encuentra el estado de los dos canales de salida. Es decir, su volumen, la amplitud del sonido y el indicador para alertar cuando hay riesgo de que las muestras saturen.

4.10. Otros módulos de audio

A continuación, se detalla el funcionamiento de algunos módulos que realizan operaciones habituales en el procesamiento de sonido, y por ello están incluidos en varios componentes descritos anteriormente.

4.10.1. *AudioDimmer*

AudioDimmer es un atenuador, reduce el volumen de las muestras. Se utiliza para ajustar el volumen del audio, en efectos como el trémolo o también en el *overdub* de los *loops* grabados.

Es puramente combinacional para que su integración sea más flexible, tal que si es preciso segmentar, pueda hacerse acorde a la arquitectura del módulo que lo contenga.

Internamente realiza la atenuación mediante una multiplicación en punto fijo (Q2.4), por un factor entre cero y uno. Una alternativa más simple y que requiere menos recursos sería desplazar las muestras hacia la derecha, para dividir entre dos la amplitud de onda. Sin embargo la falta de precisión de este método, unido a la amplia capacidad de las FPGAs actuales, hace que la primera opción sea mucho más conveniente.

4.10.2. *AudioBooster*

Este módulo funciona bajo el mismo principio que *audioDimmer*. La diferencia reside en el factor de multiplicación, que al ser mayor que uno amplifica el volumen de las muestras. Por este motivo, incluye además un control de desbordamiento, el cual devuelve el máximo valor representable (positivo o negativo) cuando las muestras resultantes exceden el límite impuesto por los 32 bits de anchura.

Se emplea, por ejemplo, en el efecto *overdrive* para incrementar el volumen de las muestras que han sido limitadas.

4.10.3. *OutlevelIndicator*

Cuando es necesario medir en qué rango de amplitud se encuentra una muestra, se utiliza *outlevelIndicator*. Es el caso de los indicadores que muestran la intensidad a la que suenan las pistas o los canales de salida en el máster. Se emplea también en la función *"autorec"* de las pistas, para detectar cuándo está sonando una entrada.

Es combinacional, por ello habitualmente se emplea en conjunto con máquinas de estados que registran su salida; cada una está adaptada al contexto en que se implementa.

Dispone de un parámetro *threshold_ step* (no es ajustable en tiempo real) para regular su sensibilidad. Con él se puede establecer una serie de 16 valores equiespaciados, que definen los rangos de amplitud para evaluar las muestras.

$$corte(i) = i * threshold\_step$$

De este modo, cuanto menor sea *threshold_ step*, más sensible será el medidor. Esto resulta útil porque en algunos casos se trabaja con amplitudes de onda relativamente pequeñas y no pueden apreciarse bien las variaciones en el volumen.

Capítulo 5

Desarrollos complementarios: carcasa y *App* para el control remoto de *LoopMAN*

Este breve capítulo está dedicado a la aplicación *software* para el control remoto de *LoopMAN* y al diseño de una carcasa que facilite conectar periféricos e instrumentos musicales al prototipo *hardware*. No obstante, el nivel de detalle no es tan profundo como en anteriores secciones ya que estos dos elementos son complementarios al objeto principal de este proyecto que recordemos es el desarrollo e implementación de la arquitectura *hardware*.

5.1. *App* para el control remoto de *LoopMAN*

Esta aplicación ha sido diseñada para su ejecución sobre el sistema operativo *iOS* y programada enteramente en el lenguaje de alto nivel *Swift*. Tiene como propósito poner a disposición una vía cómoda y flexible para controlar el *looper*. Por un lado proporciona una interfaz gráfica interactiva, por otro se encarga de enviar al sistema *hardware* los mensajes de control pertinentes y también recibir la información de su estado.

5.1.1. Funcionalidad de la aplicación

Tal y como muestra la figura 5.1, la interfaz gráfica principal presenta un panel de control. Cada uno de sus elementos está asociado a un parámetro ajustable del *looper*.

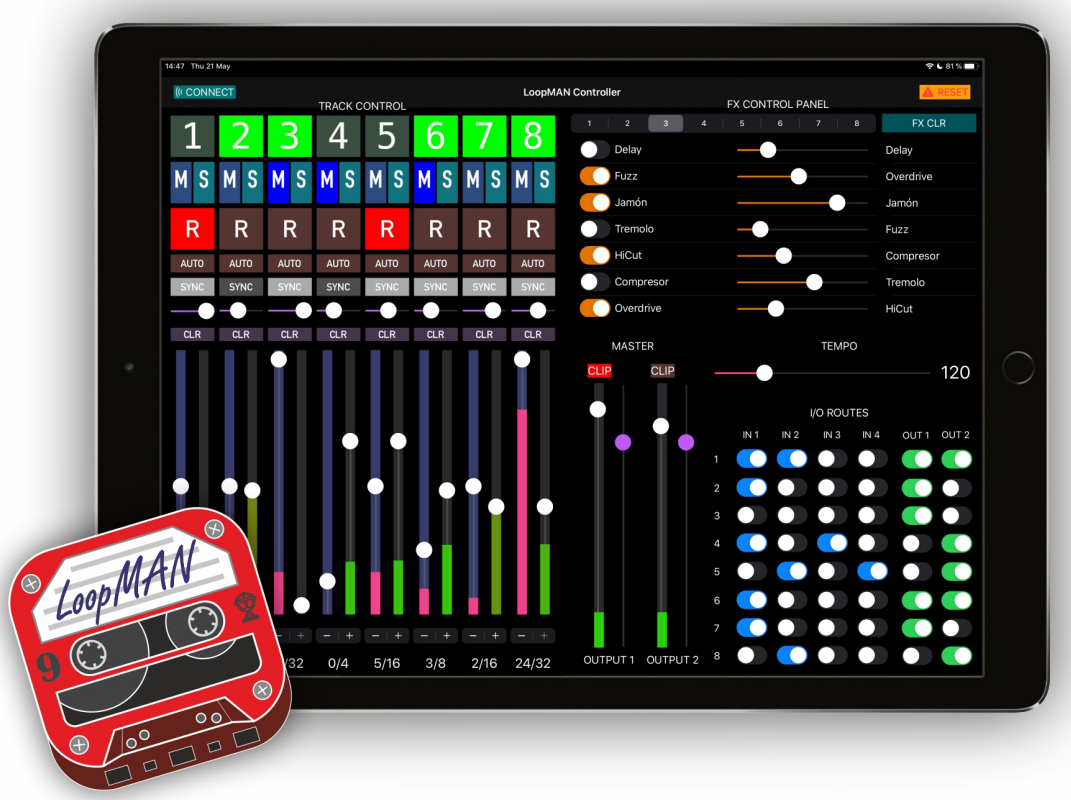


Figura 5.1: Aplicación *software* diseñada para el control.

En la mitad izquierda del panel (*TRACK CONTROL*) se observan los controles de las ocho pistas. En primer lugar en la parte superior, se sitúan los botones numerados de *play*; estos inician o detienen la reproducción del audio grabado en su pista. A continuación se encuentran *mute* y solo, para silenciar las pistas. El siguiente es el botón para iniciar o detener la grabación (*rec*), seguido del control de autograbación (comienza a grabar automáticamente, al empezar a sonar). Después está *sync*, para activar o desactivar la sincronización de la pista en cuestión. Le sigue un *slider* (barra deslizable) para ajustar el *overdub*. Finalmente se sitúa el botón que efectúa el borrado del audio de la pista, *clear*.

Inmediatamente debajo están los reguladores de volumen y de longitud del audio grabado. Ambos muestran mediante una animación la amplitud actual del sonido y avance de la reproducción, respectivamente. Además del *slider* pueden utilizarse los botones +/- para ajustar la longitud de la pista.

La mitad superior derecha del panel contiene el control de efectos (*FX CONTROL PANEL*). Este consta de una lista de selección a la derecha, para elegir los efectos a aplicar en el orden deseado; y a la izquierda otra lista que muestra los efectos seleccionados en la pista actual. Mediante la barra superior (que tiene un elemento desplazable y unos números) es posible ajustar para qué pista están siendo configurados los efectos.

Al seleccionar un efecto de dicha lista desplazable (la situada a la derecha), este será encolado en el *LoopMAN* y aparecerá en la lista de efectos (situada a la izquierda). Para limpiar la cola de efectos creada, está disponible el botón *fx clear*.

Cada efecto encolado dispone de un interruptor para activarlo o desactivarlo, así como un *slider* para regular su intensidad.

Debajo de los efectos, hay un *slider* horizontal, el cual permite ajustar el *tempo* del sistema.

La sección etiquetada como *I/O ROUTES* permite configurar a qué pistas está conectada cada entrada; así como por qué salidas suena cada pista.

Por último, en *MASTER* se sitúan cuatro deslizadores, que sirven para ajustar el volumen

y el nivel de compresión de las salidas de audio del sistema. Al igual que en las pistas, muestran la amplitud del sonido con una animación. También hay dos indicadores luminosos (*clip*) para indicar cuando el volumen de salida es demasiado alto.

5.1.2. Principios de la aplicación *software*

A continuación se detallan los rasgos principales de la arquitectura de la aplicación *software*, en base a lo expuesto sobre cómo se realiza la comunicación a través de *Bluetooth* en las secciones 4.9.2 y 4.8.2, para el envío y recepción respectivamente.

El dispositivo externo que realiza el control *Bluetooth* recibe las tramas con la información del estado y actualiza lo que muestra de forma acorde a las mismas. Podría optimizarse parte de la comunicación, puesto que si el dispositivo de control envía una trama al sistema con la petición (por ejemplo) de iniciar la grabación en una pista, el mismo dispositivo podría actualizar localmente esta información y mostrar directamente que hay una grabación en curso. Todo ello sin esperar la recepción de una trama procedente del *looper* notificando el mismo cambio. Sin embargo esto podría dar lugar a problemas de inconsistencia, ya que la información se mantendría de forma redundante. Incluso ciertos escenarios quedarían sin resolver, pues bajo determinadas circunstancias (e.g. un error) podría no producirse la respuesta esperada del sistema.

La conclusión es que la mejor forma de implementar el dispositivo externo de control por *Bluetooth* es como un mero terminal, que envía datos y muestra gráficamente la información que recibe, sin tomar ningún tipo de decisión.

Tal y como se ha expuesto en el capítulo 4, la información solo se envía o recibe cuando se producen cambios en el estado de algún módulo o se interacciona con algún control. El objetivo es minimizar consumo y uso de la red.

Como ya se ha detallado, las tramas tienen longitud fija, 24 bits. Su formato es el mismo que el descrito en el capítulo anterior (figuras 4.30 y 4.25).

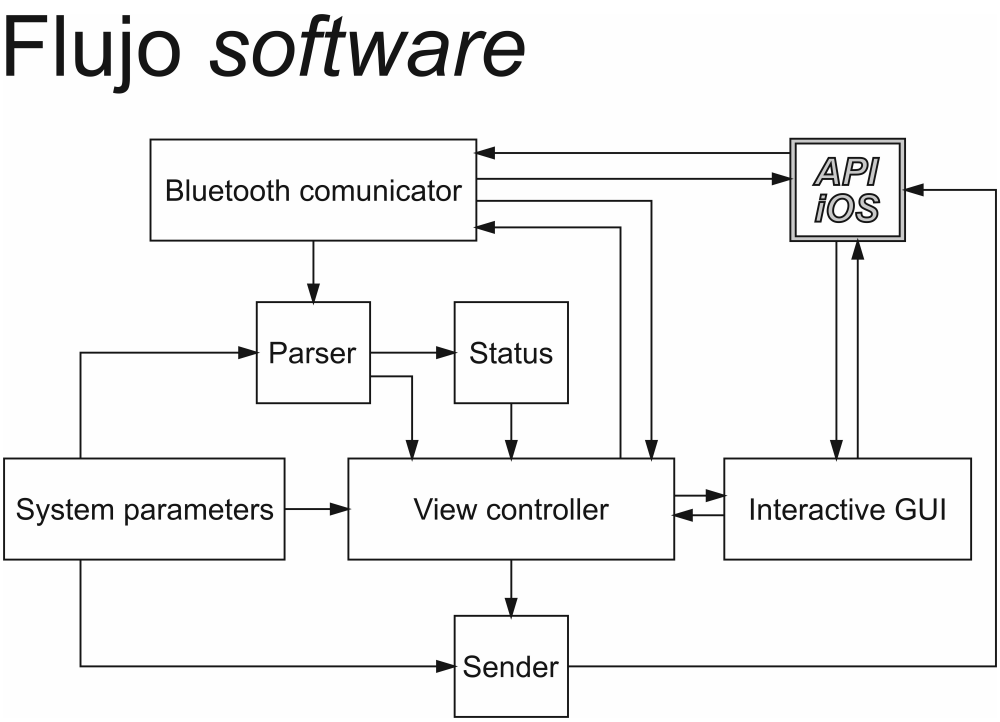


Figura 5.2: Esquema de la arquitectura *software*.

5.1.3. Arquitectura *software*

La figura 5.2 muestra un esquema general de los componentes de la aplicación.

En primer lugar y siguiendo el orden de la ejecución, se encuentra el componente *Bluetooth comunicator*, que realiza el establecimiento de la conexión con el dispositivo y actúa como monitor de tramas entrantes. Para ello comprueba los paquetes entrantes y obtiene de estos las tramas de tres bytes que envía al *Parser*, encapsuladas en una estructura de datos.

En el *Parser* se interpretan los mensajes recibidos, es decir, se extrae la información contenida en ellos. En base a esta se realizan los cambios pertinentes en el componente *Status*, que alberga el estado actual del sistema.

El controlador de la vista gráfica (*View controller*) administra el estado y comportamiento de los elementos de la interfaz de usuario. De este modo, el componente *Parser* notifica cuando se producen cambios en el estado, tal que *View controller* haga trascender los cambios a los botones, indicadores, *sliders* y demás elementos. Por otro lado, si el usua-

rio interactúa con alguno de dichos controladores, *View controller* identifica el elemento en cuestión y lee el nuevo valor. A continuación no lo almacena en *Status*, si no que construye la trama con el mensaje asociado al cambio y la envía a *Sender* para retransmitirla al *looper*. Cuando este último procese la información y aplique el cambio, retransmitirá su nuevo estado a la aplicación. Ahora sí, los cambios quedan reflejados en *Status*.

Este factor es clave en el paradigma de funcionamiento de la aplicación, que no debe procesar ni contener ningún tipo de lógica que no sea imprescindible. Simplemente actúa como un terminal que envía órdenes al componente central (el *looper* en este caso), así como recibe y muestra información del estado.

El módulo *System parameters* que aparece en el diagrama simplemente contiene el valor de las características paramétricas del *looper*: el número de efectos (y sus nombres), el número de entradas y el de salidas. Así también es posible hacer que la aplicación escale horizontalmente, acorde al *hardware* y sin ningún esfuerzo adicional. En *System parameters* también se encuentra la codificación de las tramas, o en otras palabras, a qué opción corresponde cada posible valor binario recibido.

En el repositorio https://gitlab.com/loopman-ndc/loopman-remoteapp está disponible el código escrito en *Swift*.

5.2. Carcasa de *LoopMAN*

A continuación se aborda el diseño del chasis que da soporte a todos los componentes físicos del proyecto. Estos son principalmente la placa de prototipado y los módulos.

5.2.1. Plano general

La estructura se compone de cuatro paredes laterales, una tapa superior, otra inferior y una barra con forma de asa. Las piezas se sujetan con 16 tornillos de métrica M3 y 12 tuercas, cuatro de las cuales son de presión. Las dimensiones de la caja son 142 milímetros de ancho por 115,5 de largo y 49 de alto. El grosor de las paredes es de tres milímetros,

mientras que las láminas inferior y superior son de dos milímetros.

El espacio interior de la montura está estrechamente ajustado a la placa, para reducir en medida de lo posible el tamaño final. Por ello también ha sido necesario emplear cables compatibles con el estándar PMOD para conectar los módulos externos (*Bluetooth* y audio). Normalmente dicha conexión se haría de forma directa, sin embargo esto requeriría una caja mucho más grande, implicando un mayor gasto de material, mayor complejidad en la fabricación y un producto final innecesariamente más grande.

La figura 5.3 muestra un plano general.

Los modelos tridimensionales del chasis pueden encontrarse en el repositorio `https://gitlab.com/loopman-ndc/loopman-enclosure`.

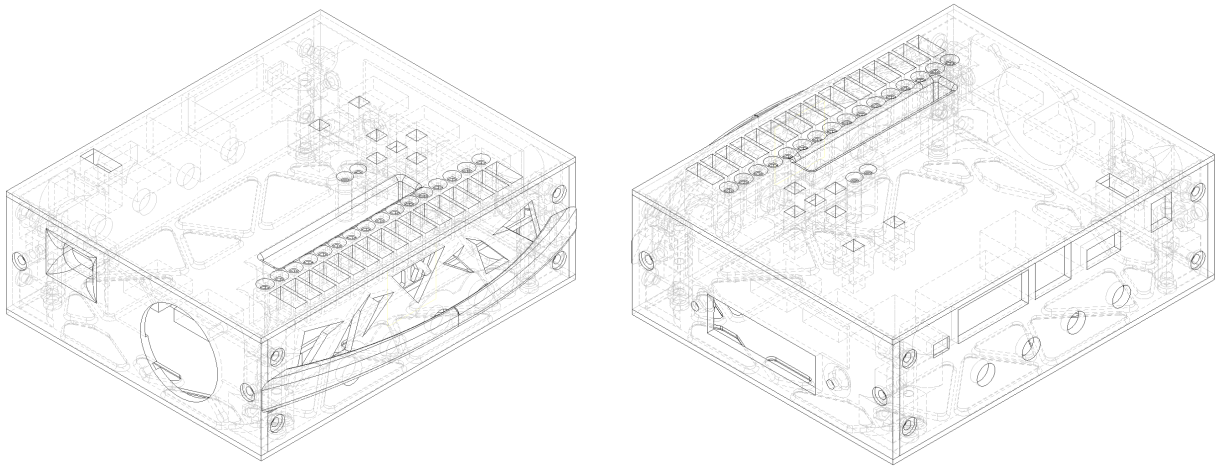


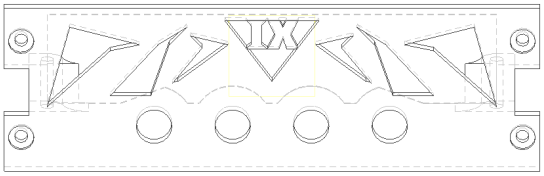
Figura 5.3: Vista general del chasis.

5.2.2. Frontal

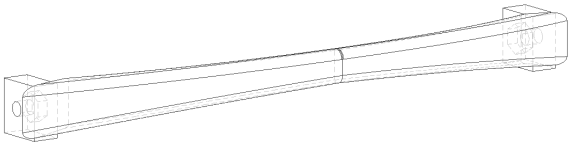
La parte frontal dispone de cuatro agujeros para fijar los conectores *jack* de 6,35 milímetros (figura 5.4), que se corresponden con las cuatro entradas de audio del sistema. La figura 5.5a muestra esta pieza.

Colocar las entradas de audio en esta parte facilita el acceso a las mismas, por si fuese necesario conectar otros instrumentos o fuentes de sonido mientras se utiliza el sistema. Por

Figura 5.4: Clavija *jack* mono de 6,35 milímetros.



(a) Frontal del chasis.



(b) Barra protectora.

Figura 5.5: Frontal del chasis y barra protectora.

ello se ha colocado una barra protectora con forma de asa, que previene golpes o movimientos accidentales en los conectores (figura 5.5b).

5.2.3. Parte trasera

Dada la disposición de los puertos en la *Nexys 4 DDR*, se han emplazado convenientemente en la parte posterior todos los orificios para acceder a los mismos (VGA y USB). También se encuentran en esta zona los conectores para las salidas de audio. En la figura 5.6 puede observarse el diseño.

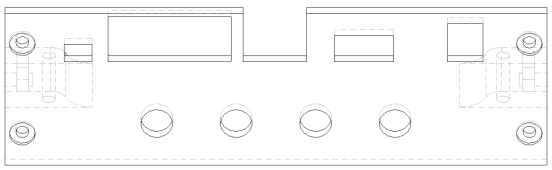
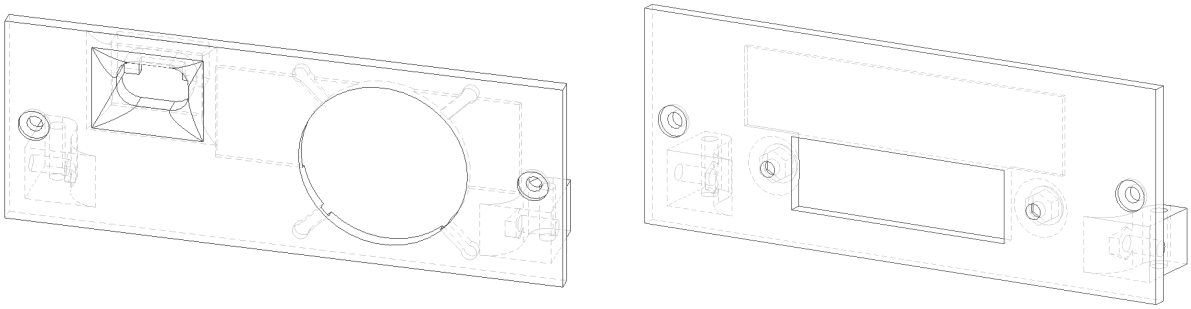


Figura 5.6: Parte posterior de la estructura.

5.2.4. Lateral izquierdo

En el lateral izquierdo se encuentra el puerto *Micro USB*, para alimentar el sistema y reprogramar en la FPGA eventuales actualizaciones del diseño. También incorpora un agujero para la refrigeración de los componentes. Se muestra en la figura 5.7a.



(a) Lateral izquierdo de la caja.

(b) Lateral derecho.

Figura 5.7: Frontal del chasis y barra protectora.

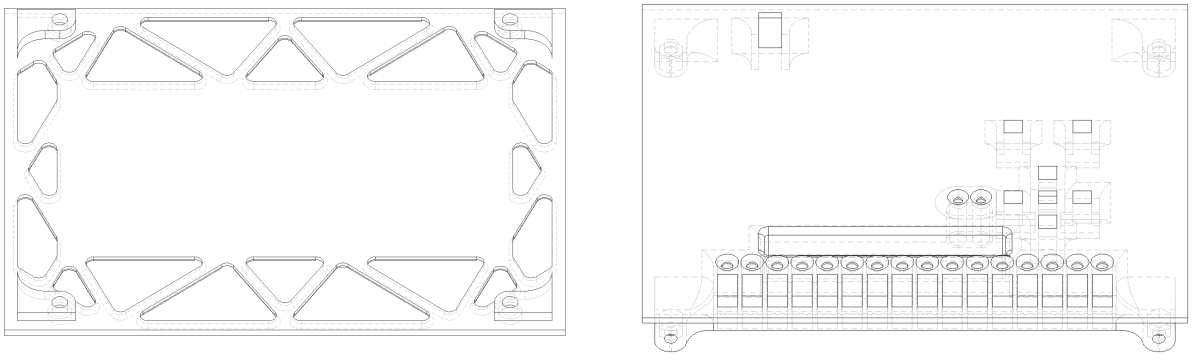
5.2.5. Lateral derecho

Esta pieza incorpora un hueco para colocar un conector *Micro ribbon* de 36 pines (figura 5.8), tal que sea posible acceder a los puertos PMOD disponibles a través de este (figura 5.7b).

5.2.6. Plataforma inferior

Sobre esta base (ilustrada en la figura 5.9a) se ubica el códec I2S.

Figura 5.8: Conector *Micro ribbon* de 36 pines.



(a) Plataforma inferior.

(b) Plataforma superior.

Figura 5.9: Plataformas inferior y superior de la estructura.

5.2.7. Plataforma superior

Finalmente la plataforma superior (figura 5.9b) se encuentra situada cerca de la placa de prototipado, permitiendo visualizar el estado de los leds y el *display* 7 segmentos, así como interactuar con los *switches* y pulsadores. Para esto último son necesarias piezas adicionales, que sirven como extensión de los controles de la placa (figura 5.10).

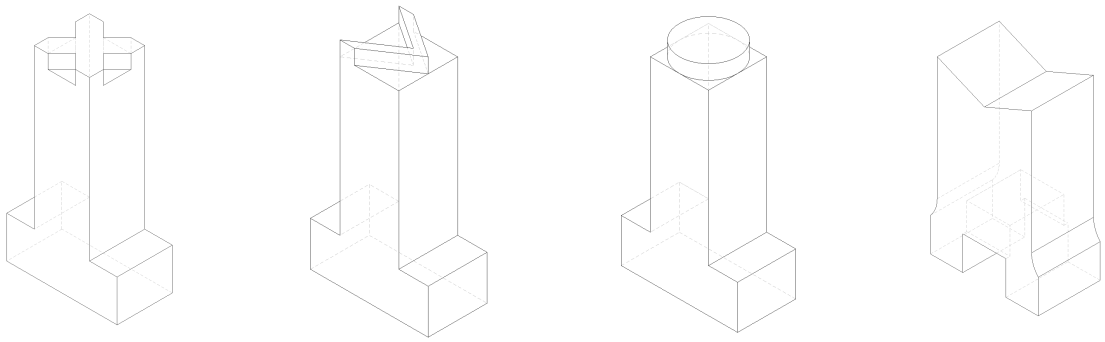


Figura 5.10: Extensiones para botones e interruptores.

5.2.8. Ensamblaje final del sistema

Para dar soporte a los distintos componentes físicos del sistema (la placa de prototipado, códecs de audio, conectores...) se procedió a la fabricación del diseño elaborado, mediante una impresora 3D. El material empleado para todas las piezas ha sido plástico ABS. La

unión de las mismas se ha realizado con tornillería de acero inoxidable.

También fueron soldados los conectores *jack* de 6,35 milímetros para la entrada/salida de audio, y el conector *Micro ribbon* para futuras extensiones.

Una vez generado el diseño *hardware* definitivo, se procedió a sintetizarlo para obtener el fichero binario de configuración de la FPGA. Este último fue escrito en la memoria *flash* QSPI que incorpora la placa, tal que al arrancar el sistema, la FPGA sea programada automáticamente con dicha configuración sin necesidad de un ordenador.

El resultado final se presenta en las colecciones de figuras 5.11 y 5.12.

Figura 5.11: *Looper* ensamblado.

Figura 5.12: Vistas del *Looper*.

Capítulo 6

Resultados, conclusiones y trabajo futuro

En este capítulo final quedan recogidos los resultados finales del proyecto, así como las conclusiones que se han obtenido en base a los mismos. También se incluye una colección de posibles vías para proseguir el desarrollo del *looper* en distintos ámbitos.

6.1. Resultados

Esta sección está dedicada a describir la consecución de los objetivos propuestos a lo largo del desarrollo de este proyecto, así como las decisiones de diseño y dificultades salvadas. También se analiza el sistema elaborado, comparándolo con otros similares disponibles en el mercado.

En el repositorio https://gitlab.com/loopman-ndc/loopman-hw puede encontrarse la descripción VHDL completa del *LoopMAN*.

6.1.1. Objetivos logrados

Desde un primer momento, las líneas generales de la arquitectura del *looper* tienden a la paralelización de todo lo posible, manteniendo una estructura y jerarquía claramente establecida. De este modo cada pista define un flujo de audio paralelo. De forma concurrente, los mecanismos de control interactúan con las pistas. Los dos objetivos perseguidos con todo

ello son el rendimiento y la escalabilidad.

Uno de los primeros hitos del desarrollo fue lograr la comunicación con la memoria DDR2 (indispensable para grabar los *loops*). Para ello se valoraron varias opciones (dada la complejidad de la tarea y la incertidumbre sobre los problemas que podría acarrear). Una de dichas alternativas fue instanciar un *Microblaze* (microprocesador) que gestionase las lecturas y escrituras. Sin embargo, finalmente se implementó mediante el *MIG*, un *IP soft core* que proporciona una interfaz de acceeso memoria. Su configuración no es trivial y exige adaptarla específicamente tanto para el módulo DDR2 externo, como para el componente interno (*multiTrackRecorder*) y los requisitos del sistema. También fue necesario conectar a dicho *IP soft core* el sensor de temperatura de la FPGA (y el conversor ADC para interpretar las lecturas).

Esta etapa del desarrollo condicionó múltiples aspectos del sistema. Entre ellos, la frecuencia del reloj principal, dependiente de la que utiliza la memoria externa; el valor de activación de los *reset* asíncronos, puesto que el *MIG* proporciona una señal *reset* sincronizada con el reloj; o la arquitectura del componente *multiTrackRecorder*, que está optimizado para funcionar con palabras de 128 bits (el máximo que permite el *hardware* utilizado).

Tras el éxito en las pruebas de acceso a memoria, fue necesario diseñar *multiTrackRecorder* tal que pudiese proporcionar en el mínimo tiempo posible los datos solicitados a memoria, teniendo en cuenta el cometido del sistema (procesar de audio en tiempo real) y las restricciones impuestas por la memoria externa y el *MIG*. Finalmente se logró que las ocho pistas pudiesen leer y escribir simultáneamente en la memoria, empleando en total (para el peor caso) dos ciclos de reloj. Esto se debe al mecanismo de cuádruple *buffer* sincronizado, que actúa como *cache* e incluye un predictor para anticiparse a las direcciones de memoria solicitadas. Dado que el objetivo era dotar al sistema de una arquitectura altamente paralela, es totalmente indispensable el acceso concurrente a memoria.

Como ya se ha mencionado, el uso del *MIG* y las características de la DDR2 obligaban a utilizar, para todo el sistema, una frecuencia de reloj determinada, en concreto de 75,

150 o 300 MHz (según la configuración). Escoger la más pequeña, ha permitido reducir el consumo, prevenir violaciones de temporización y utilizar palabras más anchas en el acceso a memoria. Sin embargo afecta negativamente al número de ciclos disponibles entre muestra y muestra para procesar el sonido, motivo por el cual todos los componentes presentes en el flujo de audio deben minimizar su tiempo de operación.

Cronológicamente, la siguiente fase consistió en desarrollar la entrada/salida de audio. Fue necesario generar los relojes I2S, considerando la restricción de los 75 MHz. Se fijó en 48828,125 Hz la frecuencia de muestreo, la más próxima al estándar de 48000 Hz (alta fidelidad) que pudo generarse a partir del reloj principal. Asimismo, desde que una muestra entra hasta que sale del sistema, resultan 1536 ciclos de reloj disponibles para el procesamiento. Utilizar una frecuencia superior habría reducido este número de ciclos, además de ser innecesario de acuerdo al teorema de Nyquist.

El retardo que induce este sistema en el sonido, es decir, el tiempo transcurrido desde que una muestra entra hasta que sale son 20 microsegundos. En comparación con sistemas basados en software considerados de baja latencia (entorno a 10 milisegundos), este *looper* es tres órdenes de magnitud más rápido.

Un detalle a recalcar es que con una plataforma de estas características, que integra tanta funcionalidad, solo hay conversores analógico-digital a la entrada y conversores digital-analógico a la salida. Mientras tanto, en un sistema basado en múltiples dispositivos digitales conectados en cadena, el audio se somete varias veces a estos procesos de conversión. Esto no solo conlleva un incremento en el retardo de la señal, sino también una paulatina pérdida de fidelidad, así como un cuello de botella impuesto por el dispositivo de peor calidad.

La resolución de las muestras enviadas los DAC y recibidas de los ADC es 24 bits. No obstante, el sistema internamente opera con el audio, sometiéndolo a infinidad de operaciones aritméticas, lo cual implica riesgos de desbordamiento. Para mitigarlo y evitar tener que saturar o truncar los datos, con vistas a preservar la calidad, los módulos internos trabajan con muestras de 32 bits.

En total, dispone de dos salidas y cuatro entradas de audio disponibles. Sin embargo, de cara al diseño, ambos valores son paramétricos y resulta inmediato ampliar dicha característica, aunque esto precisa disponer del *hardware* requerido (es decir, más audio códecs).

El mezclador de entrada se comunica directamente con el módulo que envía las muestras al PMOD I2S, por tanto en él se realiza el ensanchamiento de las muestras (previo a cualquier tipo de procesado), incorporándoles los ocho bits de guarda. Para la entrada, fue diseñado cuidadosamente el *audioMixer*, que mediante una arquitectura multiciclo emplea un solo sumador para mezclar un número paramétrico de entradas. Además incorpora un sistema para prevenir desbordamientos y aplicar saturación cuando sea irremediable. Este componente se integra dentro del *trackInputMixer*, que gestiona la asignación de canales de entrada a las pistas.

Por otro lado, para el mezclador de salida (*outputMixer*) se diseñó otro mezclador basado en el *audioMixer*: *compressorMixer*. Este incorpora un ajuste de ganancia para los canales *master* (de salida). Para evitar la saturación, dispone de un compresor que atenúa la señal cuando su amplitud es demasiado elevada, es decir, elimina la distorsión indeseada. Este último está configurado para actuar con suavidad, tal que su efecto en el sonido pase desapercibido en medida de lo posible. El algoritmo de compresión analiza las muestras después de procesarlas, para determinar automáticamente la proxima operación a aplicar sobre las mismas.

De forma complementaria a *trackInputMixer*, el módulo *outputMixer* reajusta las muestras a 24 bits para enviarlas al códec de audio externo.

El diseño de las pistas ha consistido fundamentalmente en sincronizar todos los elementos que rigen su comportamiento, en su mayoría corresponden a ajustes del usuario: reproducción, grabación, *overdub*, volumen, canales de entrada y salida... De forma paralela también funcionan automáticamente el generador de direcciones de memoria, los mecanismos de sincronización entre pistas, el mezclador, el *autorec*, etcétera. En definitiva, el espectro de situaciones posibles en las pistas es muy amplio y coordinar todos los submódulos no es en

absoluto trivial, menos aún cuando se pretende minimizar el número de ciclos empleado en procesar, pues esto exige un diseño paralelo y no secuencial. La implementación se realizó mediante múltiples FSM interdependientes, cuyas transiciones entre estados están coordinadas, tal que las máquinas (de estados) interfieran entre sí. Además de internamente en cada pista, ha sido necesario gestionar la sincronía entre las propias pistas y el controlador central de *tempo*, el *tempoGenerator*; algo esencial para camuflar el error que cometemos los humanos a la hora de pulsar un botón (concretamente, el de reproducir o grabar) con precisión en un instante de tiempo determinado. *TempoGenerator* supuso además implementar aritmética en punto flotante para el cálculo de la media y porcentajes. Esto exigió numerosas pruebas para validar y garantizar el correcto funcionamiento de las pistas, así como extensas depuraciones del *hardware* para detectar errores del diseño.

Para que los usuarios intractúen con el *LoopMAN* existen varias vías tanto para controlarlo, como para recibir información de su estado. Esto mejora la tolerancia a fallos del sistema (por redundancia). Es decir, si se produjese un error en el *tablet*, siempre se podría seguir realizando el control a través del monitor y el teclado.

El control "tradicional" mediante el teclado y el monitor exigió un conocimiento previo de los protocolos PS/2 y VGA respectivamente, para implementar los módulos de comunicación con los periféricos. Por otro lado también ha sido necesario diseñar un componente que realizase la traducción de las tramas del teclado a las señales de control internas del sistema. Para el VGA, existe un componente que dibuja los gráficos de la pantalla en base al estado de los componentes; la principal dificultad es que dicha interfaz se genera con lógica combinacional, que para cada píxel de la pantalla manifiesta el color correspondiente.

La comunicación por cable del monitor VGA y el teclado USB satisfacen con creces la velocidad de respuesta exigida, siendo perfectamente posible controlar el sistema mediante dichos periféricos

En lo que respecta al control remoto, el módulo de comunicación por *Bluetooth* permite un intercambio fluido de información con el dispositivo de control. Esto ligado a la comple-

titud de la aplicación software hace que la interacción con el sistema sea ágil, resultando inmediato realizar cualquier ajuste.

La versión del protocolo *Bluetooth* empleado es la 4.2. La razón de este hecho se debe a la programación *software* de la aplicación para *iOS*; puesto que la API proporcionada por el fabricante para acceder a los componentes del sistema no permite utilizar directamente la versión 2.1 del estándar *Bluetooth*.

La única forma de operar con versiones de la especificación anteriores a la 4.0 era mediante una licencia especial para desarrolladores de accesorios para dispositivos *iOS*, cuya obtención es absolutamente inviable para uso no profesional.

En cuanto a los detalles de la comunicación remota, se han utilizado tramas de tres bytes de anchura. Este tamaño se ha estimado el mínimo necesario para albergar toda la información necesaria para los mensajes, considerando las ocho pistas del sistema y el número de bits utilizado por los parámetros (entre cuatro y cinco bits); pero dejando margen para ampliar la funcionalidad sin que sea necesario rediseñar el protocolo.

El controlador remoto es esencialmente el *tablet* con la aplicación. El principio básico de su diseño fue crear un terminal, que muestra lo que recibe y envía mensajes de control independientes al estado actual del sistema. Es decir, que los mensajes sean equivalentes a accionar un pulsador, no un interruptor (que para "encenderlo" ha de estar apagado y viceversa). Su implementación exigió lograr la comunicación *Bluetooth*, elaborar un mecanismo de recepción y parseo de tramas, otro para la elaboración y envío de las mismas, y una GUI interactiva.

De este modo quedan cubiertos (en el plano del control) los requisitos de latencia y fiabilidad que exige este sistema de tiempo real. Además se facilita considerablemente el manejo del *looper*, a pesar de todas las funciones que ofrece.

El procesador de efectos debía ser capaz de imitar a las pedaleras típicamente utilizadas por los músicos, donde el orden de los efectos es relevante. Además, para mayor versatilidad, cada efecto ha de presentar parámetros ajustables en tiempo real. Con ambas ideas presentes

se diseñó la arquitectura de *fxController*, capaz de crear cadenas de efectos configurables.

Mientras tanto, para cada efecto desarrollado fue necesario aplicar las técnicas pertinentes de procesado digital de señales, además de simular su comportamiento en *Matlab* de forma previa a la implementación *hardware*. Desde el punto de vista técnico los efectos requieren utilizar componentes tales como *block ram*, *DSPs* o *MACs* para efectuar las operaciones aritméticas. Además, cada uno de ellos atiende a conceptos teóricos y algoritmos dispares por lo cual solo ha sido posible reutilizar entre algunos de ellos su arquitectura; el resto de las implementaciones son particulares a cada uno, incluso el tipo de representación en punto fijo (QN.M) está escogida específicamente para cada efecto.

También ha sido necesario realizar multitud de pruebas para ajustar los parámetros fijos de algunos efectos (como el compresor), y lograr que su funcionamiento sea el adecuado. Además debe tenerse en cuenta que hay factores como el error de cuantización, que igualmente puede afectar al sonido provocando resultados inesperados (que no se han contemplado en las simulaciones).

Concretamente los filtros han requerido especial dedicación, ya que actúan en el dominio de la frecuencia, lo cual complica considerablemente el procesado. Esto se debe a que los algoritmos no son lineales, es decir, que para actuar sobre la muestra actual hay que considerar las anteriores (también ocurre en los efectos *flanger* y *jamón*). Además los aspectos teóricos a considerar son más complejos, al igual que su arquitectura.

Finalmente, para la elaboración de la carcasa se procuró reducir tanto como fuese posible las dimensiones, sin sacrificar ninguna característica de la placa de prototipado (puertos, pulsadores, leds...). De este modo, se minimiza el coste del material y se obtiene un diseño más atractivo. Sobre la carcasa, se tuvo en consideración que fabricar con ABS es más complejo que con otros plásticos (en tecnología de impresión FDM), sin embargo las propiedades mecánicas y el acabado son notablemente superiores.

Para concluir este apartado, la figura 6.1 ilustra el *LoopMAN* con sus cuatro entradas conectadas a instrumentos (micrófono, bajo, guitarra y teclado), además de sus dos salidas

Relación de componentes *hardware* y sus características		
Componente	Características	Propiedades
FPGA	*Slices* Biestables por *slice* LUTs por *slice* DSP	15850 8 4 (de 6 entradas) 240
Memoria	DDR2 SDRAM *Block* RAM *Flash* QSPI	128 MiB 4860 Kbits (593,26 KiB) 16 MiB
Vídeo VGA	Profundidad de color Resolución	12 bits 640x480 píxeles
Audio Codecs IIS	Resolución de muestra Max. frecuencia de muestreo (entrada) Max. frecuencia de muestreo (salida)	24 bits 96 KHz 192 KHz
Bluetooth	Especificación compatible	4.2
USB HID	Interfaz PS/2	(No aplicable)

Tabla 6.1: Componentes *hardware* empleados en la construcción del sistema.

a un amplificador y un PA. También tiene conectados un monitor VGA, teclado y *tablet* con la *app* de control (a través del enlace *Bluetooth*).

6.1.2. Especificaciones técnicas del *looper*

En el cuadro resumen 6.1 se especifican las características técnicas de los componentes utilizados en la construcción del sistema.

En la tabla 6.2 se resumen las características finales del *LoopMAN*.

Los datos referidos al consumo de energía (marcados con un asterisco) son valores estimados, puesto que debido al confinamiento no ha sido posible acceder al equipamiento necesario para realizar una medida precisa. La potencia disipada por la FPGA se ha obtenido de la herramienta de síntesis (*Vivado*), mientras que las referidas a componentes *hardware* externos proceden de sus *datasheets*.

Figura 6.1: El *LoopMAN* en todo su esplendor.

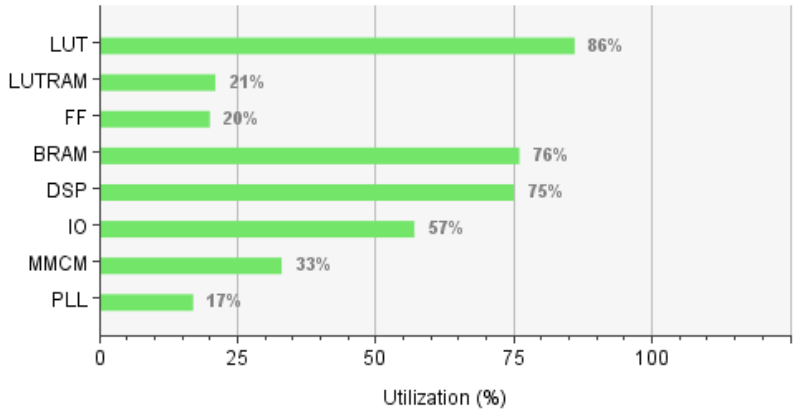
Especificaciones técnicas del *looper*		
Sistema	Frecuencia de reloj	75 MHz
	Alimentación	5V (Micro USB)
	Tiempo de cómputo máx. (por muestra)	1119 ciclos
	Potencia disipada (FPGA)	1,56 W *
	Potencia disipada (SDRAM)	324 mW *
	Dimensiones	14,2 x 11,55 x 4,9 cm
Audio	Resolución de muestra	24 bit (32 bit internamente)
	Frecuencia de muestreo	48828,125 Hz
	Retardo inducido	20 μs
	Entradas de audio	4
	Salidas de audio	2
	Tipo de conexión	Jack 6.35 mm
	Potencia disipada (audio códecs)	80 mW *
Looper	Número de pistas	8
	Tiempo de grabación	5,72 minutos
	Máximo número de compases	32
	Tempo	45-400 bpm
	Overdub	Ilimitado
	Autograbación	Sí
Efectos	Número de efectos	9
	Cadena de efectos configurable	Sí
	Delay	0-0,168 segundos
	Jamón	0-0,168 segundos
	Compresor	-
	Fuzz	-
	Overdrive	-
	Flanger	0-2,62 milisegundos
	Trémolo	0,04-1,34 segundos
	HiCut	15k, 3,5k, 1k, 400, 220 Hz
	LowCut	25, 40, 80, 140, 190 Hz
Vídeo	Puerto VGA	Resolución 640x480 píxeles
Control	Puerto USB	Compatible con teclados USB HID
	Potencia disipada (USB HID)	100 mW *
	Bluetooth	Versión 4.2
	Potencia disipada (*Bluetooth*)	43 mW *

Tabla 6.2: Especificaciones del *LoopMAN*.

6.1.3. Utilización de recursos

Una vez completada en su totalidad la descripción *hardware* (en VHDL) del sistema, la herramienta de síntesis (*Vivado 2018.2*) obtuvo los resultados expuestos en esta sección.

En lo que respecta al uso de recursos de la FPGA (figura 6.2), se observa un alto uso de memoria BRAM (block ram), debido a los efectos basados en retardos y los filtros FIR. Ambos almacenan fragmentos de audio con 32 bits de resolución.



Resource	Utilization	Available	Utilization %
LUT	54727	63400	86.32
LUTRAM	3953	19000	20.81
FF	24943	126800	19.67
BRAM	103	135	76.30
DSP	180	240	75.00
IO	119	210	56.67
MMCM	2	6	33.33
PLL	1	6	16.67

Figura 6.2: Utilización de recursos de la FPGA.

También es elevado el porcentaje de DSPs empleados, dado su uso en los mezcladores que operan en punto fijo, así como en determinados efectos (donde hay presencia de aritmética moderadamente compleja).

Los *buffers* de acceso a la memoria externa, y especialmente la forma de almacenar los coeficientes de filtrado (en los efectos) disparan el número de LUTs necesarios por encima

Coste del prototipo			
Componente	Precio	Cantidad	Total
Digilent Nexys 4 DDR	235 €	1 ud.	235 €
Digilent PMOD I2S2	20 €	2 ud.	40 €
Digilent PMOD BLE	22 €	1 ud.	22 €
Conector Jack 6,35 mm	1,5 €	8 ud.	12 €
Conector Micro ribbon 36 pines	2,65 €	1 ud.	2,65 €
Tornillos allen M3 8 mm	18 €/50 ud.	10 ud.	3,6 €
Tornillos allen M3 14 mm	20 €/50 ud.	4 ud.	1,6 €
Tornillos philips M3 40 mm	12 €/50 ud.	4 ud.	0,96 €
Tuercas hexagonales M3	8 €/100 ud.	18 ud.	1,44 €
Filamento ABS negro	25 €/Kg	200 g	5 €
Total			324,25 €

Tabla 6.3: Coste económico de los componentes del *LoopMAN*.

de los cincuenta mil. Sin embargo en lo que respecta a la lógica, a pesar de su complejidad, el diseño no presenta grandes estructuras matriciales ni nada similar. Por ello el margen restante es moderadamente amplio como para introducir nuevos módulos en el diseño.

El uso de biestables es también considerable, pero la capacidad de la FPGA empleada es mucho mayor. Al igual que sucede con los LUTs, la arquitectura tampoco se compone de grandes bancos de registros ni se asemeja, por ejemplo, a la estructura de las GPUs.

Los pines de entrada/salida quedan ocupados por los códecs de audio, la interfaz Bluetooth, puertos VGA, USB y los pulsadores.

6.1.4. Coste económico del prototipo

En cuanto al coste final de los componentes y materiales empleados en la elaboración del *LoopMAN* es necesario considerar por un lado los componentes físicos: PMODs, placa de prototipado, cables y conectores; y por otro los materiales empleados en la fabricación de la carcasa (plástico y tornillería).

La tabla 6.3 contiene una relación del coste de dichos componentees y materiales.

Figura 6.3: *Boss RC-300 Loop Station*.

6.1.5. Comparativa con otras implementaciones

Ya en la sección dedicada a los antecedentes de este trabajo (apartado 1.3), se mencionaron varias implementaciones comerciales de *loopers*.

Pese a que no existe un equivalente en términos tecnológicos (puesto que ninguno está diseñado puramente en *hardware*), hay unidades cuya funcionalidad es bastante similar.

Modelos de gama alta

Un ejemplo dentro del ámbito profesional es el *Boss RC-300 Loop Station* (figura 6.3). En cuanto a especificaciones técnicas quedaría por detrás de este sistema en varios aspectos como el número de pistas (tres respecto a ocho), resolución de las muestras (16 bit/44100 Hz respecto a 24 bit/48800 Hz). Este modelo, al contrario que el diseñado, no dispone de mecanismos para gestionar múltiples entradas y salidas con flexibilidad. No obstante, hace posible almacenar tres horas de *loops*, mientras que el aquí presentado apenas ofrece seis minutos. Otra ventaja del *Boss RC-300* es que sus pistas son estéreo.

Sobre la interacción con el usuario, el *RC-300* no dispone de conexión a monitor de vídeo, ni control mediante teclado USB y tampoco tiene interfaz Bluetooth. Sin embargo, sí es capaz de sincronizarse con otros dispositivos vía MIDI.

A día de hoy (junio de 2020) precio de este *looper* multipista ronda los 460 euros[1].

[1] Precios obtenidos de la web `https://www.thomann.de`.

Modelos de gama media

También existen alternativas más económicas en el mercado de los *loopers* multipista. Ejemplos son el *tc electronic Ditto X2 Looper* o el *Pigtronix Infinity Looper 2* (figura 6.4).

(a) *tc electronic Ditto X2 Looper*.

(b) *Pigtronix Infinity Looper 2*.

Figura 6.4: *Loopers* de gama media.

Estos modelos no suelen presentar más de dos pistas grabables y tampoco disponen de múltiples entradas. No pueden aplicar efectos al sonido. Además no disponen de ningún tipo de control más allá de los *switches* que hay en ellos, y la información de estado se muestra mediante unos leds.

Solo realizan las funciones básicas de grabación (con *overdub*) y reproducción. Las dos pistas se sincronizan entre ellas. Para los modelos mencionados el tiempo máximo de grabación es de cinco minutos.

Como se observa, los *loopers* de esta categoría son mucho más simples que los de gama alta, y carece de sentido compararlos con el sistema diseñado.

Su precio varía en función del fabricante, para el *tc electronic Ditto X2 Looper* se sitúa entorno a los 150 euros; en el caso del *Pigtronix Infinity Looper 2* la cifra asciende hasta los 250 euros[2].

[2]Precios obtenidos de la web https://www.thomann.de.

6.2. Conclusiones

En base a la implementación realizada y los resultados expuestos en la sección anterior, se han obtenido las siguientes conclusiones.

6.2.1. Viabilidad del diseño *hardware*

Elaborar un diseño puramente *hardware* exige a quien lo desarrolla un mayor esfuerzo que el *software*. El primero es mucho más "delicado", no puede tomarse una decisión de implementación sin considerar el contexto, todo ocurre en paralelo. Ni siquiera es trivial realizar una suma y almacenar su resultado. Tampoco existen grandes bibliotecas como las que hay en el mundo del *software*, con funciones para resolver casi cualquier problema sin apenas programar.

Asimismo, depurar y verificar un diseño *hardware* requiere mucha mayor dedicación.

Otro detalle, es que en *hardware* la descripción (en un lenguaje HDL) puede ser correcta, pero no válida si el diseño resultante no cumple las restricciones establecidas.

No obstante, esto no significa que los diseños *hardware* sean inviables o carezcan de sentido fuera de los chips de propósito general (microprocesadores o microcontroladores).

FPGAs

Gracias a la tecnología de las FPGA, es posible paliar la mayor parte de las dificultades anteriores. Ya en la sección 2.2 se detallaron los hechos que motivaron a la elección de esta plataforma para el desarrollo, y ahora puede concluirse que la decisión ha sido acertada.

Se han probado una infinidad de prototipos en distintas etapas del desarrollo, la depuración de los mismos se ha realizado con un IP *soft core*: el *Integrated Logic Analizer*. De este modo es perfectamente viable corregir y validar los diseños *hardware*, a pesar de que en cualquier caso es más complejo y tedioso que en *software*. [21]

Por otro lado las actuales herramientas de automatización de diseño electrónico (o por sus siglas en inglés, EDA) enfocadas a FPGAs hacen triviales los procesos de síntesis y

place and route, tal que todo el esfuerzo de quien diseña puede centrarse en la especificación RTL. Como ya se ha mencionado anteriormente, se ha empleado el lenguaje VHDL para la descripción y *Vivado 2018.2* como herramienta EDA.

Sin embargo, con el computador doméstico utilizado para ejecutar la herramienta[3], esta puede llegar invertir entorno a 45 minutos en proporcionar resultados para el diseño completo con módulos de depuración incluidos. Lo cual también entorpece el proceso de pruebas y corrección de errores, especialmente en las últimas etapas del desarrollo.

En definitiva, realizar un diseño *hardware* de propósito específico, es perfectamente viable hoy en día, tanto dentro como fuera del ámbito profesional o de investigación. Las herramientas de diseño y las FPGAs de la última década contribuyen enormemente a dicha viabilidad, no obstante sigue siendo bastante más complejo que el *software*.

6.2.2. Ventajas del *hardware*

En las secciones previas se han mencionado diversas desventajas que supone realizar un diseño puramente *hardware* frente a uno *software* (obviando las opciones mixtas). Sin embargo, en este apartado se resume el muy amplio abanico de beneficios que aporta.

En términos de rendimiento, todo son ventajas. El grado de optimización es máximo, pues cada componente es diseñado específicamente para tareas concretas, lo cual se traduce en velocidades más altas la hora procesar los datos.

Este hecho puede sorprender, teniendo en cuenta que las frecuencias de reloj en las FPGAs son notablemente más bajas de lo que se acostumbra a ver en microprocesadores. Motivo por el cual, también se logra un menor consumo.

Además la paralelización de operaciones es absoluta e inherente a la naturaleza del *hardware*, el volumen de trabajo puede ser verdaderamente inmenso en la solución implementada.

Si la arquitectura se construye con cuidado, como ha sido el caso de este *looper*, los diseños gozan de una escalabilidad excelente (en gran medida debido la naturaleza paralela antes comentada).

[3]Se ha utilizado un computador convencional de sobremesa, de gama media-alta.

Quizá no siempre sea necesario disponer de toda la potencia que aporta el *hardware* puro, y no en todos los escenarios merece la pena hacer el esfuerzo que requiere este tipo de implementación. Pero cuando el rendimiento y la estabilidad son imprescindibles, por ejemplo en los sistemas de tiempo real, probablemente no exista solución más idónea que un diseño *hardware*. Es importante mencionar también que el alto rendimiento y la capacidad para escalar, ligados a la tecnología del *hardware* dinámicamente reconfigurable, hacen que los sistemas de este tipo resistan muy bien al paso del tiempo.

6.3. Trabajo futuro

A lo largo de esta sección se listan una serie de posibles vías para continuar con el desarrollo del sistema.

6.3.1. En el ámbito técnico

Tal y como se mostraba en los resultados, especialmente al comparar con otras implementaciones de *loopers* de gama alta, uno de los inconvenientes de este sistema es la poca memoria disponible para las grabaciones.

Una posible solución sería migrar el proyecto a otro modelo de placa de prototipado, con un módulo RAM de mayor capacidad. Alternativamente podría valorarse emplear otro medio de almacenamiento, por ejemplo uno persistente que permita guardar los fragmentos grabados de forma permanente.

También puede considerarse actualizar la herramienta de síntesis (*Vivado*) así como los IP *cores* utilizados, para aprovechar las mejoras de las versiones más recientes. Probablemente esto requeriría realizar ciertos ajustes en el diseño.

6.3.2. En el ámbito arquitectónico

Escalar el sistema

Con vistas a evaluar la escalabilidad del sistema, podría probarse a extender tanto como fuese posible aspectos tales como el número de entradas y salidas, de efectos o de pistas.

De cara al código sería inmediato para los tres primeros, puesto que dichos valores son paramétricos. Para las pistas serían necesarias más modificaciones, puesto que están fuertemente ligadas al acceso a memoria.

Reducir el tiempo de ciclo

Otra mejora a realizar podría ser incrementar la frecuencia de reloj, con vistas a aumentar la cantidad de ciclos que hay disponibles entre muestra y muestra para realizar el procesamiento del audio. Esto requeriría segmentar algunos caminos, para evitar violaciones de temporización; no obstante, la medida no es en absoluto contraproducente, puesto que no serían demasiados, solo para cierta aritmética.

Pipeline funcional

Una última propuesta en este ámbito sería adaptar el flujo principal de audio a un *pipeline* funcional cuyas etapas son multiciclo. Aunque los cambios a realizar son mínimos, porque en realidad es muy cercano a la implementación actual, consistiría en eliminar la restricción de sacar una muestra cada vez que entra la siguiente. De este modo cada componente dispondría de más ciclos para el procesado de audio, lo que supone una mejora en la escalabilidad del sistema.

Sobre esto hay varias consideraciones a tener en cuenta, en primer lugar, que podría perjudicar al retardo inducido (aunque como es tan pequeño seguiría siendo inapreciable). Y por otro lado, que no supondría ninguna mejora palpable de cara al resultado final, puesto que el sistema actual no lo necesita (ya que todo el procesamiento "cabe" holgadamente en los 1536 ciclos disponibles).

Figura 6.5: Controlador MIDI *Livid DS1*.

6.3.3. En el ámbito funcional

Interfaz MIDI

Como se comentó en los resultados (sección 6.1.5) es frecuente que los *loopers* de gama alta dispongan de interfaces MIDI para sincronizarse con otros instrumentos electrónicos. Otra posibilidad sería ir un paso más e implementar también compatibilidad para controladores MIDI (figura 6.5), que proporcionan un medio físico similar a la aplicación software diseñada para el control del *looper*.

Ecualizador multibanda

Por ejemplo, entre el procesador de efectos y el mezclador de salida, podría implementarse un ecualizador gráfico[4] multibanda para cada pista. Es muy frecuente el uso de este tipo de dispositivos en el contexto de la producción musical, y permitiría ajustar el sonido y realzar

[4]Los ecualizadores de tipo gráfico permiten amplificar o atenuar rangos de frecuencias prefijadas (conocidos como "bandas").

u ocultar ciertos matices (por ejemplo imperfecciones o armónicos indeseados).

Una propuesta de implementación es mediante un banco de filtros FIR con frecuencias de corte acotadas.

Efectos basados en transformadas de Fourier discretas

Aunque ya se han implementado algunos efectos basados en el dominio de la frecuencia (los filtros paso baja), podrían incorporarse al diseño otros nuevos más complejos e interesantes.

Para ello sería preciso implementar el algoritmo FFT (transformada rápida de Fourier), así como su inversa para reconstruir el audio. También sería necesario realizar convoluciones entre los datos.

Debe tenerse en cuenta que incorporar esto al diseño requiere muchos recursos *hardware*.

Posibles resultados serían filtros de alta resolución, puertas de ruido, *vocoders*, etcétera.

Bibliografía

[1] Udo Zolzer. *DAFX: Digital Audio Effects*. John Wiley & Sons, Ltd., 2002.

[2] Carlos Aznar Ruiz, Germán Ramos Peinado, and Rafael Gadea Girones. Trabajo de Fin de Grado: Diseño de Aceleradores Hardware para Procesado de Audio en Arquitecturas SoC. *Universidad Politécnica de Valencia*, 2016.

[3] Math Verstraelen, Jan Kuper, and Gerard J.M. Smit. Declaratively Programmable Ultra Low-Latency Audio Effects Processing on FPGA. 2014.

[4] Sujit Rokka Chhetri, Bikash Poudel, Sandesh Ghimire, Shaswot Shresthamali, and Dinesh Kumar Sharma. Implementation of Audio Effect Generator in FPGA. *Nepal Journal of Science and Technology*, 15(1):89–98, 2014.

[5] Julius Orion Smith. *Physical Audio Signal Processing*. http://ccrma.stanford.edu/~jos/pasp/, 2010.

[6] Les Thede. *Practical Analog and Digital Filter Design*. Artech House, 2004.

[7] Steve Winder. *Analog and Digital Filter Design*. Newnes, 2002.

[8] Xilinx Inc. *Artix-7 FPGAs Data Sheet*.

[9] Integrated Silicon Solution Inc. *IS43/46DR16640C datasheet*.

[10] IBM Corp. *Video Subsystem, Section 2. VGA Function*.

[11] USB Implementers' Forum. *Device Class Definition for Human Interface Devices (HID)*.

[12] Adam Chapweske. The ps/2 mouse/keyboard protocol. https://isdaman.com/alsos/hardware/mouse/ps2interface.htm, 2001.

[13] Digilent Inc. *PMOD BT2 Reference Manual.*

[14] Microchip Technology Inc. *Bluetooth 4.2 Low Energy Module.*

[15] Microchip Technology Inc. *RN4870/71 Bluetooth Low Energy Module.*

[16] NXP Semiconductors. *UM10204 I2C-bus specification and user manual.*

[17] Cirrus Logic Inc. *CS4344/5/8 Stereo D/A Converter.*

[18] Cirrus Logic Inc. *CS5343/4 Multi-Bit Audio A/D Converter.*

[19] Xilinx Inc. *Zynq-7000 SoC and 7 Series Devices Memory Interface Solutions.*

[20] Xilinx Inc. *Clock Generator (v4.03a).*

[21] Xilinx Inc. *Integrated Logic Analyzer LogiCORE IP Product Guide.*

[22] Peter J. Ashenden. *The designer's guide to VHDL.* Elsevier Science & Technology, 2008.

[23] Eric Tarr. *Hack Audio: An Introduction to Computer Programming and Digital Signal Processing in MATLAB.* Routledge, 2019.

[24] William C. Pirkle. *Designing audio effect plug-ins in C++ : with digital audio signal processing theory.* Focal Press, 2013.

[25] S. Yao. Audio effect units in mobile devices for electric musical instruments. *IEEE Access*, 7:159239–159250, 2019.

[26] Tammy Noergaard. *Embedded Systems Architecture*, chapter 4. Elsevier, 2005.

[27] Digilent Inc. *Nexys4DDR FPGA Board Reference Manual.*

Apéndice A

Introduction

A.1. Motivation

Over the last one hundred and fifty years, music has undergone a series of constant changes and innovations in all respects. Regarding the compositions, new genres have emerged, which have been gradually fusing between them. There is not the slightest doubt that the stylistic variety is boundless.

Moreover, the continuous appearance of new instruments and techniques has allowed to broaden the horizons of musical compositions enormously. The entrance of the digital world in our lives has an effect on all that surrounds us, and music is not an exception.

Traditional instruments, which could be considered analogical, are at the mercy of the performer; are subject to our imperfections. Such defects are regarded as beautiful, they turn the sound into something natural with the randomness that it implies. However, to create music with computers and generate synthetic acoustic waves has made easier that anyone can develop their creativity, delegating to a machine the skills and knowledge required by instruments. To sound on the radio is no longer necessary a band of three, four or all those that are needed; armed with guitars and bass guitars. It is only enough a computer and maybe a microphone.

Here is the point where this project intercedes: one single person making music in real time, without having previous knowledge; combining both analogical and digital worlds.

To sum up, the idea is to be able to run the *live looping*[1] technique using one single device that incorporates the very necessary functionality so one sole person may sound as a whole band.

Nevertheless, the name of this bachelor final project is *Hardware Looper*: there are not only reasons to create a looper, but also to make it in hardware.

Implementing the system in an FPGA allows to obtain a much higher output than would be achieved with software on microprocessors or microcontrollers, and with a less cost. In addition, the fact that the FPGA hardware is dynamically reconfigurable makes possible to update the design in the future.

It is also important to emphasize that the hardware is inherently parallel, which makes easy to maintain high-performance systems without adversely affecting its output.

To create a dedicated hardware design is therefore a guarantee for the future, although it requires a bigger initial investment in terms of work and effort.

A.2. Objectives

The main goal of this project is to design and implement a device on an FPGA, to which connect the audio output of digital or analogical instruments. It should be able to record pieces of music to play them later in loop.

The system must also mix these inputs (as well as the outputs) permitting all kind of combinations in order to reach more versatility. In respect to the sound, this must be processed with the objective of applying diverse effects to the output.

The union of both analogical and digital worlds is not trivial and even less when is about recorded pieces, since a human cannot determine the duration of these with absolutely accuracy. This leads to alignment and synchronization problems of the recordings that must be resolved when executing specific mechanisms.

[1] In the field of the musical performance, the technique known as *live looping* consists in the recording and later playback of musical pieces (loops) in real time. This method emerged in the mid-nineties.

Because of the multitude of functions and all configurable options, there should be produced an easy-to-use resource for monitoring and displaying the machine state.

With respect to the design, system's architecture must be modular and parallel as well as present a good scalability. The facts that motivate an implementation in pure hardware are in section 2.2.

A.3. Background

The variety of papers similar to this looper is certainly not so wide, however there are some aimed at the implementation in hardware of effects processors in real time. Research articles can be also found along the same lines. [2–4]

These papers and research agree on their conclusions regarding the high processing speed, that is reached due to the use of FPGAs, and on the good scalability of this kind of platform.

Beyond the academic world exists a great range of comercial devices with common purposes such as those in this project (figure A.1). From a functional point of view it is nevertheless not usual to find multiple audio inputs and outputs in these devices. Although some are able to incorporate effects to the audio.

From a technical perspective all these products use microcontrollers to run software that defines its behaviour. In fact, in terms of digital audio processing, there is only one company that commercialize devices with implementations in dynamically reconfigurable hardware: *Antelope Audio, Inc.* They are effects processors, as the one in the figure A.2. This kind of product is only focused on professional environments where the highest performance is required and the budgets are significantly wide.

A.4. Work plan

The development of this project has spanned from September 2019 to June 2020. For the organization it has been considered appropriate to establish a series of sequential stages, giving priority to those that more dependencies caused. Such phases are listed in order

(a) *Ditto X4*.

(b) *Boss RC-505*.

(c) *Boss RC-300*.

Figura A.1: Multitrack loopers.

Figura A.2: Digital effects processor with FPGA (model *Discrete 8 Synergy Core*).

below:

1. **Project statement.**

   In this stage prior to the development, the nature of the system was specified: a looper implemented in hardware on an FPGA. Despite the fact that the project would not start a few months later, it was necessary to do the enrollment.

2. **Analysis of the selected platform and assessment of the deployment options.**

(Two weeks).

Once the prototype board was selected, some design restrictions were determined due to its components; such as audio samples' width or video's color depth. Likewise, the functional details of the system began to be to settled: recording in multiple tracks, adjustable effects, several audio inputs and outputs, etcetera.

3. **Development of a memory controller DDR2 adapted to the prototype board.** *(Six weeks).*

   According to the conclusions of the previous stage, it was decided to begin with a purely hardware implementation of the memory controller. On this component depends the sound recording, which is essential in a looper.

   During this stage, on the basis of a previous controller, a new one was elaborated, adjusted to system requirements, using a suitable clock rate and the highest allowed word width. This task required configuring an *IP soft core* besides, which works as a memory interface.

   Once it was feasible to write and read all memory addresses, several architectures were suggested in order to provide a solution for the problems of concurrent access, data alignment and access times. When the controller was finally designed, it was implemented in hardware.

   After the success in its verification and bug fixing, the idea of a *hardware-software* co-design is dismissed (which would make memory access easier, by implementing the controller in software).

4. **General approach of the architecture.** *(One week).*

   As soon as the DDR2 controller was resolved, parameters like system's clock rate or the number of tracks were set. The outlines of the design, its hierarchy and communication principles between modules were also determined. Moreover, it was necessary

to fix: which components were essential and which tasks must these make to fulfill functionality.

5. **Design of the I2S controller.** *(One week).*

   The I2S controller is required for the audio input-output in the system, to validate all modules that sound goes through.

   From a controller designed to operate at 75 MHz, a new one was made compatible with the multiple inputs and outputs mechanism.

   Having completed the implementation thereof, it was feasible to test recording and playback capacities of the previously designed memory controller.

6. **Design of the input and output mixers.** *(Two weeks).*

   During this stage, the modules that mix system's multiple inputs and outputs were planned, according to the proposed architecture. It was necessary to design a parametric system able to dynamically establish all posible route combinations among audio inputs, tracks and outputs. Different problems which arise when mixing audio (such as saturation) were also handled through control mechanisms.

7. **Tracks and synchronization.** *(Four weeks).*

   Tracks implementation meant a link between the audio input/output and the memory access.

   During this development stage the control logic was made as well as synchronization and coordination mechanisms between tracks, since in these last, the recordings are managed. In addition, it was essential to design the memory address generator (for the communication with the DDR2 controller), which is also subject to the synchronization.

   The metronome (device for main *tempo* control in the system) was implemented too.

8. **VGA and PS/2 controllers integration.** *(Two weeks).*

   Such controllers are essential for testing the tracks, so they were implemented right after. The procedure consisted not only in adapting a video controller in such a way that is compatible with the used prototype board, but also in generating combinational structures the bitmaps (stored in ROMs), which are necessary to print the information graphically on the external monitor.

   Regarding the keyboard, the translation processes of PS/2 messages to internal signals were defined for controlling the looper.

   Both controllers were encapsulated in components that concentrate in them the state transmission and control reception tasks. That way it is possible to include other human interface devices, so they can work simultaneously.

   At this stage of the development, an initial version of the system was achieved.

9. **Design of *Bluetooth* control modules.** *(Three weeks).*

   The next goal was to design the modules for *Bluetooth* communication. That implied establishing a protocol and format for the sending and reception frames. However, it was first verified that communication was possible between both devices. There were incompatibility problems, so it was necessary to change the *Bluetooth* module.

   Given that the response time is critical, it was considered appropriate to minimize as much as possible the amount of exchanged information, despite the negative effect in the complexity. In this way, sendings or receivings are only performed when some parameter's value changes, instead of transmitting the whole system state periodically.

   For the receiver, it was necessary a frame parser to produce the relevant control signals. Nevertheless, for the emisor it was required to adapt the system to *Bluetooth* transmissions slowness, as they imply multiple clock cycles. A labels mechanism with enqueuable frames was designed for this purpose.

10. **Software implementation of the remote control app.** *(Three weeks).*

    To control the system via *Bluetooth*, it was developed an app aimed at mobile devices with *iOS* operating system. During this stage, a software architecture was devised, able to establish communication for data sending, reception and representation.

    Received events are interpreted, their information is stored and displayed on an animated graphical user interface. This GUI is also interactive: it monitors user's actions, in order to generate control frames and send them to the looper.

11. **Multi-effects processor design.** *(Four weeks).*

    The first step of this stage was to design the mechanism that allows the user to select in which order the effects should be applied.

    Effect's design demanded firstly to make software implementations of these and the corresponding sound checks, in order to implement in hardware the respective architectures to these effects.

    The effects have architectures that differ from each other: some use memories, different arithmetic operators, combinational logic elements...

12. **Incorporation of filters.** *(Two weeks).*

    With the objective of completing the effects collection, some filters were designed and incorporated into the fx processor. It was essential to research into these, in order to choose the most appropriate algorithms to implement them later. Just like with other effects, software simulations were previously run.

13. **Elaboration of an enclosure for the looper.** *(Two weeks).*

    Finally, an enclosure was created using computer-aided design tools (CAD), in order to assemble all components of the product. To materialize this chassis, it was used a 3D printer with fused deposition modeling technology.

14. **Variable cut frequency in filters.** *(One week).*

    It was decided to improve filters implementation, adapting them so it was possible to vary the cutoff frequency. After evaluating several options, it was time to complete the design.

A.5. Organization of the project

The rest of this project is arranged in series of sections with the following contents:

- **Overall view of the LoopMAN system**: here is described what it is and what can the looper do, as well as the general guidelines of its internal structure. This part also includes the reasons that have motivated to use an FPGA as platform.
- **Technological context**: it compiles all resources that have been essential to accomplish this project. Also contains technical information about system components, protocols, standards, implemented algorithms and used tools.
- **Hardware architecture and LoopMAN's implementation**: details the hardware implementation of the components in which the system is subdivided; in addition to their internal modules and how these are communicated. That includes the elements of the principal audio flux, as well as control and synchronization mechanisms, and management of interaction between the user and the looper.
- **Supplementary development: enclosure and App for the remote control of the *LoopMAN***: this part deals briefly with the functionality that the app offers and its internal architecture. It includes the characteristics of the chassis too, which is designed to keep the system's components.
- **Results, conclusions and future project**: Features of the final system, besides of everything that has been concluded during the development of this bachelor final project. Different ways to keep developing this project are also presented in this section.

- **Bibliography**: list of source materials that are referred to in this report (articles, books, previous works...).

Apéndice B

Conclusions

Final *LoopMAN* specifications are summarized in the table B.1.

Power consumption related values (marked with an asterisk) are estimated, due to lockdown it has not been possible to properly measure them, as required tools could not be accessed.

FPGA dissipated power has been obtained from the same tool used for synthesis (*Vivado*), while the values of external hardware components have been retrieved from their datasheets.

Based on the implementation and obtained results, the following has been concluded.

B.1. Hardware design feasibility

Elaborating a purely hardware design demands, whom develops it, a greater effort rather than software. The first is much more "delicate", it cannot be implemented without considering the context; as everything occurs in parallel. To perform an addition and store the result is not even a trivial matter. It is not the case that there are big libraries as those in the world of software, with functions to solve almost any problem without hardly programming.

Likewise, to debug and verify a hardware design needs much more dedication.

Other detail is that the hardware description (in HDL language) may be correct, but not valid if the resultant design does not fulfill the established constraints.

***LoopMAN's* technical specifications**		
Sistema	Clock frequency	75 MHz
	Power	5V (Micro USB)
	Max processing time (per sample)	1119 cycles
	Dissipated power (FPGA)	1,56 W *
	Dissipated power (SDRAM)	324 mW *
	Dimensions	14,2 x 11,55 x 4,9 cm
Audio	Sample resolution	24 bit (internally 32 bit)
	Sampling frequency	48828,125 Hz
	Delay	20 μs
	Audio inputs	4
	Audio outputs	2
	Port	Jack 6.35 mm
	Dissipated power (audio códecs)	80 mW *
Looper	Number of tracks	8
	Recording time	5,72 minutes
	Max number of beats	32
	Tempo	45-400 bpm
	Overdub	Unlimited
	Autorecording	Sí
Effects	Number of effects	9
	Configurable effects chain	Sí
	Delay	0-0,168 seconds
	Jamón	0-0,168 seconds
	Compresor	-
	Fuzz	-
	Overdrive	-
	Flanger	0-2,62 miliseconds
	Tremolo	0,04-1,34 seconds
	HiCut	15k, 3,5k, 1k, 400, 220 Hz
	LowCut	25, 40, 80, 140, 190 Hz
Video	VGA port	Resolution 640x480 pixels
Control	USB port	USB HID keyboard compatible
	Dissipated power (USB HID)	100 mW *
	Bluetooth	Version 4.2
	Dissipated power (*Bluetooth*)	43 mW *

Tabla B.1: *LoopMAN's* specifications.

However, this does not imply that hardware designs are not feasible or even nonsense apart from general-purpose chips (microprocessors or microcontrollers).

B.1.1. FPGAs

Thanks to FPGAs technology, is possible to mitigate most of the previous difficulties. There in the section 2.2 were concreted in detail the reasons that motivated to choose this platform for the development, and now can be concluded that the decision has been right.

A huge number of prototypes were tested in different stages of the development, whose debugging has been made with an IP soft core: the *Integrated Logic Analizer*. In this way, to debug and validate hardware designs is completely possible, despite the fact that in any case is more complex and tedious than in software. [21]

Furthermore, the current electronic design automation tools (EDA) aimed at FPGAs make banal both synthesis and "place and route" processes, so that the designer's whole effort can be focused on the RTL specification. As previously mentioned, VHDL language has been used for the description and *Vivado 2018.2* as EDA tool.

Nevertheless, with the home computer used to run the tool[1], it spends approximately 45 minutes until providing results for the entire design with depuration modules included. This also retards testing and bug fixing processes, specially in the last development stages.

In conclusion, making a special-purpose hardware design is totally feasible nowadays, both inside and outside of professional and research areas. The design tools and FPGAs from the last decade have helped towards the success of such feasibility, however it is still much more complex than software.

B.2. Hardware advantages

Along previous sections, multiple disadvatages of purely hardware designs over software ones have been mentioned (mixed options apart). Furthermore, this section summarizes the

[1]It has been used a conventional, mid-to-high end, desktop computer.

very wide range of hardware benefits that hardware provides.

About performance, its all gain. Optimization level is maximum, as every single component is specifically designed for each task. This means much higher speeds when it comes to process data.

This fact may be surprising, considering that clock rates in FPGAs are usually quite slower than microprocessors. That is why lower power consumptions are also achieved.

Moreover, hardware operations are totally and inherently parallel by default, so huge workloads can be processed on implemented designs.

Whether architecture is carefully built, as it has been in this looper, designs are well scalable (also due to its previously mentioned parallel character).

Pure hardware implementations are extremely powerful, and might not be always worth to do the extra effort that designing requires. Although, when performance and stability are mandatory (for example in real time systems), there is probably no better way to implement it than with a hardware design. It is also important to mention that high both performance and scalability tied to reconfigurable computing technology, make this kind of systems future-proof.

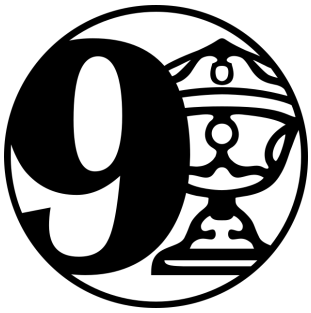
LoopMAN

Septiembre 2019 - Junio 2020

Entre coronavirus y otras calamidades.

Imposible sin Mendi y Carlos.

Diseñado en San Rafael por



nueve de copas